SESSION

FORMAL METHODS + PROCESS AND PRODUCT LINE IMPROVEMENT + AGILE DEVELOPMENT AND MANAGEMENT + WORKFLOW MANAGEMENT + PROJECT MANAGEMENT ISSUES

Chair(s)

TBA

Formal Modelling in the Concept Phase of Product Development

Mathijs Schuts[1], **and Jozef Hooman**[2,3]
[1]Philips HealthTech, Best, The Netherlands
[2]Embedded Systems Innovation (ESI) by TNO, Eindhoven, The Netherlands
[3]Radboud University, Nijmegen, The Netherlands

Abstract— *The traditional process framework for product realisation in industry often leads to a long and difficult integration phase. An important reason is that in the concept phase only informal descriptions are made about the required product, its decomposition, and the interfaces between components. We propose a formal modelling approach for the concept phase, using a new light-weight modelling tool to formalize system behaviour, decomposition and interfaces. The confidence in the product concept is increased by simulation, both manual and automatic with random system characteristics. By means of a dedicated graphical user interface, communication with different stakeholders is improved. We discuss the application of the proposed approach at Philips HealthTech.*

Keywords: Formal models; software engineering; concept definition; simulation

1. Introduction

We propose a method to improve the concept phase of product realisation by means of formal techniques. A traditional development process from concept to a validated product is depicted in Figure 1, see for instance [1]. It describes six distinct phases between concept and product. During the concept phase an informal document is being created with a high level description of the concept. This document is reviewed and agreed upon by all stakeholders. The document consists of a decomposition of the developed product, the different hardware and software components it consists of, the responsibilities per component, and the interaction between the components, possibly with an informal interface description. From the concept description, different development groups concurrently start developing the component they are responsible for. This may also include 3rd party components developed by other companies.

Such a process framework provides a structured way to come from concept to product and allows the concept to be decomposed into different components such that multiple development groups can concurrently work on the different components. A frequently occurring problem in industry, however, is that the integration and validation phase takes a large amount of time and is rather uncontrollable because many problems are detected in this phase and might require a redesign of components.

An important reason for these problems is the informal nature of the concept phase. Clearly, this leads to ambiguities and inconsistencies. Moreover, only a part of the complete behaviour is described in an informal document, often only a part of the basic functional behaviour without taking errors or non-functional aspects into account. The complete behaviour is defined during the implementation phase of the different components. Hence, a large part of system behaviour is implicitly defined during the implementation phase. If multiple development groups work in parallel in realizing the concept, the integration phase can take a lot of time because the independently developed components do not work together seamlessly. Another problem is that during the integration phase sometimes issues are found in which hardware is involved. Then it is usually too late to change the hardware and a workaround in software has to be found.

To prevent these types of problems, we propose the use of formal modelling techniques in the concepts phase, because it is early in the process and all consecutive phases can benefit from an improved unambiguous concept description. Moreover, errors made in this phase are very costly to repair in a later phase [2], [3].

By making a formal model of the system in the concept phase, ambiguities, contradictions and errors are removed from the informal concept description. During modelling one is forced to think about the exceptional behaviour early in the development process. Many questions needs to be answered which would be implicitly defined during the implementation phase otherwise. Moreover, by formalizing interface descriptions, less problems during the integration phase are expected. Figure 2 depicts a graphical representation of the proposed extension of the product realisation framework.

The formal model is developed incrementally to allow updates after aligning with stakeholders and to incorporate new insights frequently. Before choosing a formal method, we first list the aspects that are important in the concept phase:

- The definition of complete system behaviour, including error scenarios.
- A clear and unambiguous definition of interfaces and concepts to support parallel development in subsequent phases.
- The possibilities to explore concepts and design decisions fast.
- Communication with stakeholders to obtain agreement on the concepts and externally visible behaviour of the product.
- The possibility to deal with a combination of hardware and software components.

Furthermore, the formal method should be easy to use by industrial engineers and scalable to large and complex systems. Based on earlier experiences, see, e.g., [4], we decided not to aim for exhaustive model checking. Since our applications consist of many asynchronous components with queues and also timing aspects are important, one almost immediately runs into state-space explosion problems.

As an alternative to increase the confidence in the system model, we will use simulation. Formal models are expressed using the Parallel Object Oriented Specification Language (POOSL). The language is supported by a simulator and a new Eclipse Integrated Development Environment (IDE). The tooling can easily be combined with a dedicated graphical user interface to support communication with all stakeholders.

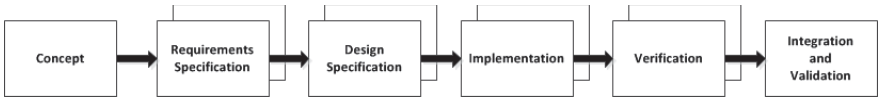


Fig. 1: Traditional process framework

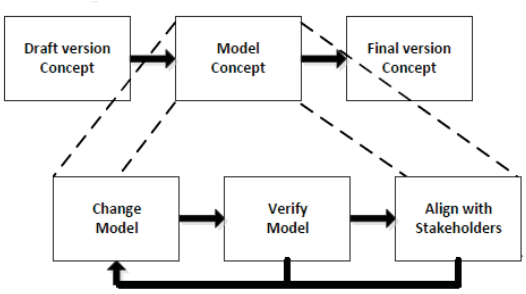


Fig. 2: Model-based concept phase

The use of formal techniques in the concept phase of hardware development has been proposed in [5]. The approach uses ACL2 logic [6] for the specification of the communication structure of a system on chip. Formal proofs of desirable properties, e.g., messages reach their destination, show the correctness of the specifications.

The application of formal methods early in the development process was already described in [7]. It describes the application of tools such as PVS [8] to requirements modelling for spacecraft fault protection systems. Although the specification language of PVS appears to be easy understandable by engineers, the interactive proof of properties is far from trivial. Hence, the conclusion of [7] proposes a rapid prototyping approach, where prototypes are tested against high level objectives.

The difficulty to use formal methods early in the development process, when there are many uncertainties and information changes rapidly is also observed in [9]. They investigated the use of formal simulations based on rewriting logic, namely Maude executable specifications [10]. The approach has been applied to the design of a new security protocol.

The paper is organised as follows. More details about POOSL and tool support can be found in Section 2. Section 3 describes the application at Philips HealthTech where the proposed method has been used. The models made for this application are presented in Section 4. Section 5 contains our concluding remarks.

2. POOSL

The long-term goal of the POOSL tooling is to shorten the development time of complex high-tech systems by providing a light-weight modelling and simulation approach. It is targeted at the early phases of system development, where requirements might not yet be very clear and many decisions have to be taken about the structure of the system, the responsibilities and behaviour of the components, and their interaction.

The approach fills a gap between expensive commercial modelling tools (like MATLAB [11] and Rational Rhapsody [12]) that require detailed modelling, often close to the level of code, and drawing tools (such as Visio and UML drawing tools) that do not allow simulation. More related to the POOSL approach is the OMG specification called the Semantics of a Foundational Subset for Executable UML Models (fUML) [13] with, e.g., the Cameo Simulation Toolkit [14].

In Section 2.1 we introduce the POOSL modelling language and describe the available tool support in Section 2.2.

2.1 POOSL modelling language

POOSL is a modelling language for systems that include both software and digital hardware. It is not intended for continuous aspects, e.g., modelling physical processes by differential equations is not possible. POOSL is an object-oriented modelling language with the following aspects:

- *Concurrent parallel processes* A system consists of a number of parallel processes. A process is an instance of a process class which describes the behaviour of the process by means of an imperative language. A process has a number of ports for message-based communication with its environment.
- *Hierarchical structure* A number of processes can be grouped into a cluster. A cluster is an instance of a cluster class which has a number of external ports and specifies how the ports of its processes are connected.
- *System definition* A system is defined by a number of instances of processes and clusters and the connections between the ports of its instances.
- *Synchronization* Processes communicate by synchronous message passing along ports, similar to CSP [15] and CCS [16]. That is, both sender and receiver of a message have to wait until a corresponding communication statement is ready to execute. A process may contain parallel statements which communicate by shared memory.
- *Timing* Progress of time can be represented by statements of the form *delay(d)*. It postpones the execution of the process by *d* time units. All other statements do not take time. Delay statements are only executed if no other statement can be executed.
- *Object-oriented data structures* Processes may use data objects that are instances of data classes. Data objects are passive sequential entities which can be created dynamically. A number of structures are predefined, such as set, queue, stack, array, matrix, etc.
- *Stochastic behaviour* The language supports stochastic distribution functions; a large number of standard distribution functions are predefined, such as DiscreteUniform, Exponential, Normal, and Weibull.

The formal semantics of POOSL has been defined in [17] by means of a probabilistic structural operational semantics for the process layer and a probabilistic denotational semantics for the data layer.

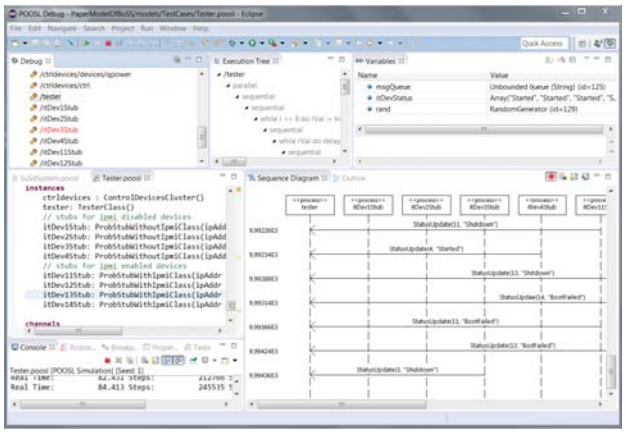
2.2 POOSL tooling

As explained in [17], the operational semantics of POOSL has been implemented in a high-speed simulation engine called Rotalumis. It supports the Software/Hardware Engineering (SHE) methodology [18]. The tool SHESim [19] is intended for editing POOSL models and validating them by interactive simulation. Recently, a modern Eclipse IDE has been developed on top of an improved Rotalumis simulation engine. The combination of the last two tools have been used for the application described in this paper.

The Eclipse IDE is free available [20] and supports advanced textual editing with early validation and extensive model debugging

possibilities. It is easy to use for industrial users and scalable to large systems; it is possible to define and simulate systems with hundreds of components. The tool contains on-line explanation and documentation.

Model validation is convenient to detect modelling errors early, before they appear during simulation. It includes checks on undeclared variables and ports, types, unconnected ports, and mismatches between send and receive statements. The debugging view shown below allows step-wise execution of models, inspection of variables, setting of breakpoints, and a running sequence diagram during simulation.



3. Application at Philips

The proposed approach has been applied at Philips HealthTech, in the context of the innovation of interventional X-ray systems. These systems are intended for minimally invasive treatment of mainly cardiac and vascular diseases. The system provides advanced X-ray images to guide the physician through the arteries of the patient to the point of interest and to execute a certain medical procedure, such as placing a stent. For a new product release, we have created a new concept for starting up and shutting down the system. This section briefly describes the informal concepts of the new start-up/shut-down (SU/SD) behaviour.

An interventional X-ray system contains a number of IT devices such as computers and touch screen modules. All IT devices can communicate with each other via an internal Ethernet control network. The IT devices are configured in such a way that they immediately start-up once they are powered. There is a central SU/SD controller which coordinates SU/SD scenarios. A user of the system can initiate a SU/SD scenario by pressing a button on the User Interface (UI). The SU/SD controller will then instruct the power distribution component to switch power taps on or off and send notification messages to the various IT devices over the internal Ethernet control network. Another scenario can be initiated by the Uninterruptable Power Supply (UPS), for instance, when mains power source fails or when mains power recovers.

The system is partitioned into two segments: A and B (for reasons of confidentiality, some aspects have been renamed). This partitioning is mainly used in the case of a power failure. When all segments are powered and the mains power is lost, the UPS takes over. Once this happens, the A segment is shut down in a controlled way, leaving the B segment powered by the battery of the UPS. If the battery energy level of the UPS becomes critical, also the B segment is shut down in a controlled way. Usually, the diesel generator of the hospital will provide power before this happens. An IT device is part of either the A segment or the B segment.

The new SU/SD concept uses the Intelligent Platform Management Interface (IPMI) [21], a standard interface to manage and monitor IT devices in a network. The IT devices in our system are either IPMI enabled or IPMI disabled.

- IPMI disabled IT devices are started and stopped directly by switching the power tap on or off.
- IPMI enabled IT devices are on a power tap that is continuously powered. To start-up these IT devices, the SU/SD controller sends a command via IPMI to them.

Combined with the two types of segments, this leads to four types of IT devices, as depicted in Figure 3.

This figure also shows that there are several communication mechanisms between the components

- Power lines for turning the power on and off.
- Control lines to connect the controller to the UI and the UPS.
- The internal Ethernet network, which is used for different purposes:
  - By the IT devices, to request the SU/SD state of the SU/SD controller and to receive SU/SD notification messages from this controller.
  - By the SU/SD controller, to ping the Operating System (OS) of an IPMI disabled IT device to observe its shut down.
  - By the SU/SD controller, to turn on an IPMI enabled IT device and to observe the shut down of the device.

A mains disconnector switch (MDS) can be used to power the complete system. An example of a SU/SD scenario is the shut-down scenario. When all segments are powered and the SU/SD controller detects that the AllSegmentOff button is pressed by the user, it will send an AllSegmentOff-pressed notification to all registered IT devices. Next all IT devices go through the following shut-down phases:

- The applications and services running on the IT device are stopped.
- The IPMI disabled IT devices will register themselves and ask the SU/SD controller to observe their shut-down. This is needed because the controller does not know which IPMI disabled devices are connected to a power tap. The IPMI enabled devices are known to the controller by configuration.
- Once the applications and services are stopped, the OS will be shut down.

The scenario ends when the SU/SD controller has detected that all IT devices are shut down. IPMI disabled IT devices are pinged to observe that they are shut down and IPMI enabled IT devices are requested for their state via IPMI to detect that they are shut down. Next the SU/SD controller will instruct the power distribution component to turn off the switchable power taps with which the IPMI disabled IT devices are powered. The IT tap that powers the IPMI enabled IT devices remains powered while these devices are in the standby state.

In the past, an abstract model of the current start-up and shut-down concept for a simpler version of the system has been made for three model checkers: mCRL2 [22], FDR2 [23] and CADP [24]. For reasons of comparison, exactly the same model was made for all three tools, leading to 78,088,550 states and 122,354,296 transitions. Model checking such a model easily takes hours. The new concept described here is far more complex because of the many asynchronous IT devices that all exhibit different behaviour. For example, the IT devices can sometimes fail to start-up or shut down. Also the timing and order in which they start-up and shut down might be different. Hence, the new concept is too complex to model check. Consequently, we decided to model the system in POOSL and used simulation to increase the confidence in the concepts.

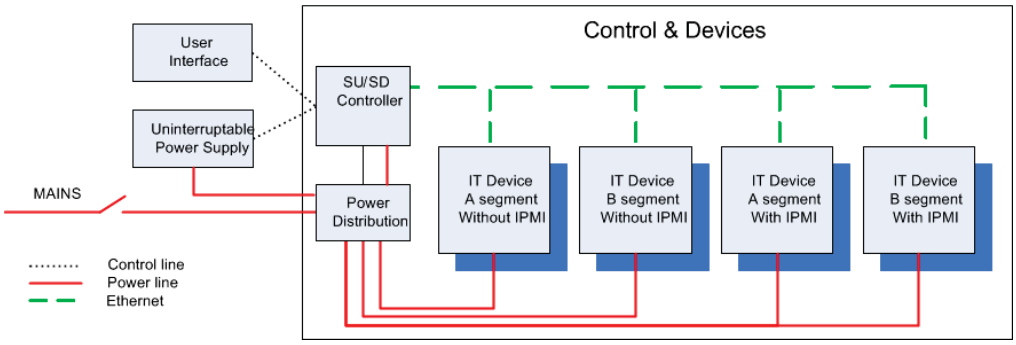


Fig. 3: System overview

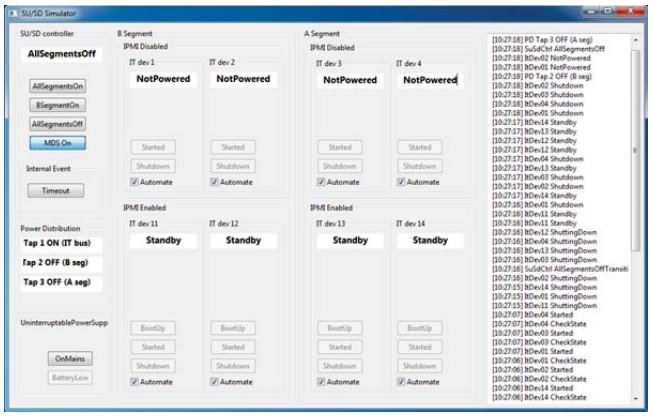
4. Modeling the SU/SD Concept in POOSL

This section describes the incremental approach to model the SU/SD concepts in POOSL. The scope of the model and the simulation environment is described in Section 4.1. Section 4.2 contains the modelling steps. A few details of the POOSL models can be found in Section 4.3. Our approach to test models automatically is presented in Section 4.4.

4.1 Modelling Scope and Simulator

The aim was to model the Control & Devices part of Figure 3 in POOSL. Besides the SU/SD Controller and the Power Distribution, the model should contain all four types of IT devices, i.e., all combinations of segments (A and B) and IPMI support. Moreover, to capture as much as possible of the timing and ordering behaviour, we decided to include two instances of each type.

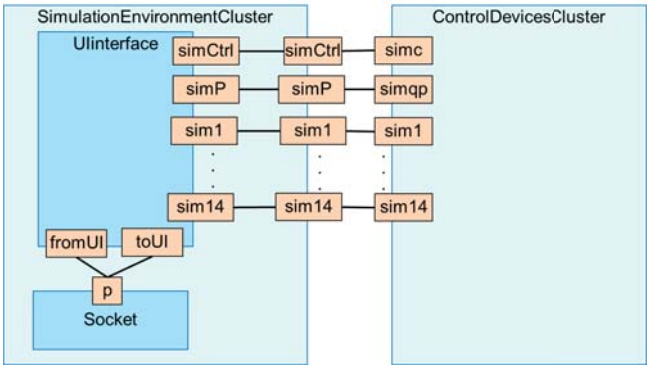
To be able to discuss the main concepts to stakeholders, we connect the POOSL model to a simulation of the environment of the Control & Devices part. We created a Simulator in Java with the use of WindowBuilder in Eclipse to allow the manual execution of scenarios. It allows sending commands from the User Interface and power components to the model and displaying information received from the model. Additionally, one can observe the status of IT devices and even influence the behaviour of these devices, e.g., to validate scenarios in which one or more IT devices do not start-up or shut down properly. The next figure shows a screenshot of the SU/SD simulator.



There are three main columns:

- The left column contains three parts:
  - On the top, the state and the UI buttons to control the SU/SD controller are displayed.
  - In the middle, the tap state of the segments is displayed.
  - On the bottom, the UPS triggers are displayed.
- The middle part contains a column for the B segment and one for the A segment; each contains a row for the IPMI disabled IT devices and one for the IPMI enabled IT devices. For each IT device the state is displayed. The start-up and shut-down behaviour of an IT device can be simulated automatically or it can be set to manual to simulate error scenarios, where the system might fall into a Timeout (see the Internal Event in the column of the SU/SD controller).
- In the right column, the status updates of the model are displayed.

The Java simulation is connected to POOSL by means of a socket. The structure of the POOSL system model is depicted in the next figure.



The system part to be modelled (the Control & Devices part) is represented by cluster ControlDevicesCluster. It has 10 external ports, one to communicate with the SU/SD controller (simc), one for power commands (simqp) and 8 for the IT devices: sim1, sim2, sim3, and sim4 for IPMI disabled devices; sim11, sim12, sim13, sim14 for IPMI enabled devices. These ports are connected to corresponding ports of the SimulationEnvironmentCluster. This cluster contains an instance of the standard Socket class provided by the POOSL library. Class UIinterface is responsible for the translation between strings of the socket interface and the SU/SD system interface.

4.2 Modelling steps

After the simulator was build, the ControlDevicesCluster has gradually been defined in POOSL. The proposed framework defines

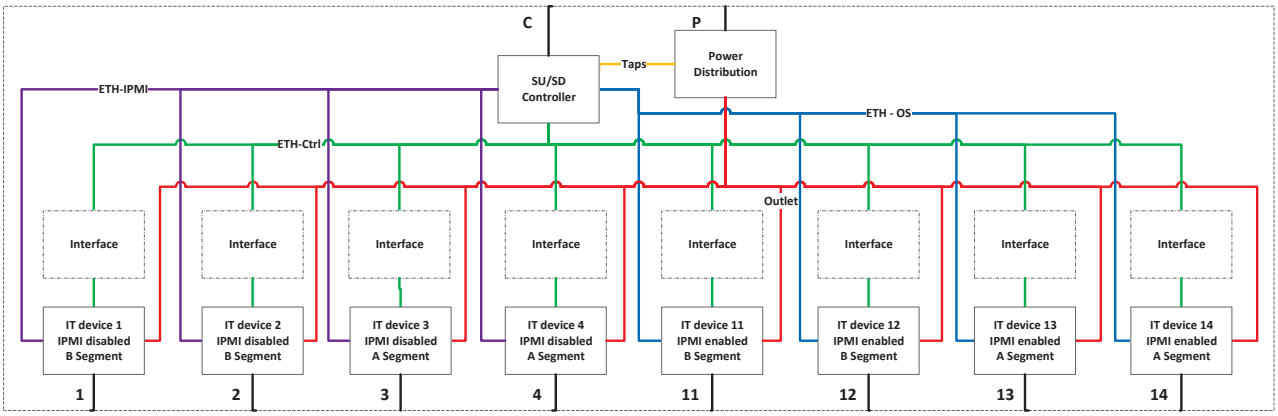


Fig. 4: Structure of the POOSL model of the ControlDevicesCluster

an incremental approach to build the model of the concept. We have used the simulator to validate the intermediate models and align the behaviour with internal stakeholders.

We started with a model of an IPMI disabled IT device and a model of the SU/SD controller for shutting down these IT devices of the A segment. In this model there were two instantiations of IPMI disabled IT devices. Note that POOSL supports a partial model where not all ports are used.

This model has been extended gradually to a model where all 8 instances of IT devices are present. Next, the SU/SD controller was extended with error behaviour to verify, for instance, that the system is always in a defined state after shut-down, which is an important requirement.

Finally, we added a model of the interface between the IT device and the SU/SD controller, because these two components will be developed concurrently. Hence, it is important to specify this contract formally and to verify it. Every IT device has an instance of the same interface model, which is implemented in such a way that the system will deadlock if the formal interface is violated. Hence, interface compliance is verified continuously during simulation.

The structure of the resulting model of this incremental approach is depicted in Figure 4.

4.3 Modelling Devices and Control

This section provides some details of the POOSL models. The first part of the model of an IT device with IPMI is shown below. It imports a library which, e.g., defines queues. Next the process class is defined, including two parameters for the IP address and the segment. All IT devices have an IP address to be able to connect them to the same network. Subsequently, the ports, the messages (only one is shown here), the variables and the initial method are defined. Note that the variables define two queues.

In the initial method *init()()*, the queues are initialized, which are FIFO by default. Next the method defines three parallel activities. The first activity defines a state machine, where the states are represented by methods. It starts the state machine by calling the initial state *ItDevNotPowered()()*.

```
import "../libraries/structures.poosl"
process class ItDevWithIpmiClass(ipAddr : Integer,
                                 segment: String)
ports
    outlet, con, ipmi, sim
messages
    outlet ? On,
variables
    msgQueue : Queue,
    ipmiQueue: Queue
init
    init()()
methods
init()() /* initial method */
    msgQueue := new(Queue);
    ipmiQueue := new(Queue);
    par
        ItDevNotPowered()()
    and
        MsgReceiveBuffer()()
    and
        IpmiReceiveBuffer()()
    rap
```

Below we show a typical definition of a state, in this case state *ItDevShuttingDown()()*.

```
ItDevShuttingDown()() | m : String |
sel
    [!(ipmiQueue isEmpty())] m := ipmiQueue remove();
    if m = "status" then ipmi ! On(ipAddr) fi;
    ItDevShuttingDown()()
or sim ? Shutdown; ItDevPowered()()
or outlet ? On; ItDevShuttingDown()()
or outlet ? Off; ItDevNotPowered()()
or sim ? Started; ItDevShuttingDown()()
or sim ? BootUp; ItDevShuttingDown()()
les
```

The state is defined as a method with local variable *m*. It selects the next state based on the contents of the *ipmiQueue* or the receipt (indicated by "*?*") of a particular message on one of its ports. Since switching a power tap on or off is instantaneous and cannot be refused by a process, all states allow the receipt of messages *On* and *Off* via port *outlet*.

The other two parallel activities of the *init()()* method are used

to model the asynchronous nature of the Ethernet communication. Method *MsgReceiveBuffer* receives messages on port *con* and stores them in queue *msgQueue*.

```
MsgReceiveBuffer()() | m : String, ip : Integer |
    con ? RecvEvent(m, ip | ip = ipAddr);
    msgQueue add(m);
    delay(1);
    MsgReceiveBuffer()()
```

Note that POOSL allows a condition on the receive statement to express that only messages with the corresponding IP address are received. Similarly, method *IpmiReceiveBuffer* stores messages in *ipmiQueue*.

4.4 Extensive Model Testing

The simulator has been used to align the behaviour with internal stakeholders and to get confidence in the correctness of the behaviour. To increase the confidence without the need of many manual mouse clicks, we created a separate test environment in POOSL. Therefore, a stub is connected to every IT device. A stub is a process which randomizes the start-up and shut-down timing of an IT device. In addition, a stub randomly decides if a device fails to start-up or shut-down. Also in these random cases the system has to respond well and it needs to be forced into defined states. The next POOSL fragment depicts how the random timing and random behaviour is implemented in the Stub.

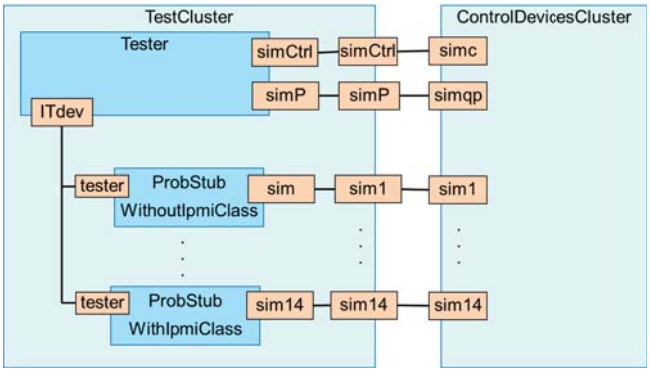
```
process class ProbStubWithoutIpmiClass
                        (ipAddr      : Integer,
                         StartUpProp : Real,
                         ShutDownProp: Real)
ports
    sim, tester
    ...

Loop()() | message : String |
[!(msgQueue isEmpty())] message := msgQueue remove();
if message = "Booting" then
    delay(rand random * 5.0);
    if rand random <= StartUpProp then
        sim ! Started;
        tester ! StatusUpdate(ipAddr, "Started")
    else
        tester ! StatusUpdate(ipAddr, "StartFailed")
    fi
fi;
```

The stubs are configured such that they fail to start-up or shut-down in 10% of the cases.

```
// stubs for ipmi disabled devices
itDev1Stub: ProbStubWithoutIpmiClass(ipAddr := 1,
                                     StartUpProp := 0.9,
                                     ShutDownProp := 0.9)
itDev2Stub: ProbStubWithoutIpmiClass(ipAddr := 2, ..
itDev3Stub: ProbStubWithoutIpmiClass(ipAddr := 3, ..
itDev4Stub: ProbStubWithoutIpmiClass(ipAddr := 4, ..
// stubs for ipmi enabled devices
itDev11Stub: ProbStubWithIpmiClass(ipAddr := 11,
                                   StartUpProp := 0.9,
                                   ShutDownProp := 0.9)
itDev12Stub: ProbStubWithIpmiClass(ipAddr := 12, ..
itDev13Stub: ProbStubWithIpmiClass(ipAddr := 13, ..
itDev14Stub: ProbStubWithIpmiClass(ipAddr := 14, ..
```

In reality the IT devices are quite reliable, but to reduce testing time it is more convenient to make the IT devices less reliable. Moreover, we are interested in the error handling behaviour of the system and not in the statistical behaviour.

For the execution of scenarios initiated by a user and the UPS, a Tester process has been created to automatically drive the system. Every stub has a feedback channel to the Tester to report the status of an IT device. The next figure depicts how the Tester and Stubs are connected to the system.



The definition of the Tester is such that it leads to a deadlock when the SU/SD controller or the IT devices do not behave as intended. Already during the first simulation run we experienced such a deadlock. The cause of the problem was found using the debug possibilities of the new POOSL IDE. We simulated the model in debug mode and inspected the sequence diagram when the deadlock occurred. In this sequence diagram we saw a problem with a message about the IPMI status of an IT device. Next we inspected the variables window shown below.

Name	Value
delayedBatteryLow	false
ipmiQueue	Empty
Occupation	0
PrimQueue	nil
QueuingPolicy	"FIFO"
Size	-1
Unbounded	-1
osQueue	Empty

It revealed that the ipmiQueue was empty, which was not expected at this point in the execution. When checking the code that handles the IPMI queue, we found that the queue was emptied after the IPMI status request has been send. The race condition was fixed by changing the order; first empty the queue and then send the IPMI status request. After fixing the race condition, the model has been executed 100 000 random start-up and shut-down cycles without experiencing a single deadlock.

5. Concluding Remarks

In the concept phase of product definition, we have used a formal system description in POOSL in combination with a graphical user interface to align stakeholders and get confidence in the behaviour of the system. We have added a model with a formal interface description between two important components of the system that will be developed concurrently. To increase the confidence in the concept, we created an automated test driver for the system with stubs that exhibit random behaviour and random timing.

While modelling, we found several issues that were not foreseen in the draft concept. We had to address issues that would otherwise have been postponed to the implementation phase and which might

easily lead to integration problems. We observed that the definition of a formal executable model of the SU/SD system required a number of design choices. We give two examples of such choices.

- If all segments are on and the UPS indicates that the mains power input fails, then the system will shut down the A segment. If, however, during this transition one or more of the IPMI enabled IT devices fail to shut down, then the SU/SD controller has no way to force these IT devices into the right state. This could be solved by an additional tap, but given the costs of an extra tap and the small chance that this will happen (both mains power and shut down of an IT device should fail), we have decided to leave it this way. If the user experiences unexpected behaviour of the system, the user can always recover the system by turning it off and on again.
- An early version of the SU/SD controller did not track if an IPMI enabled IT device did in fact start up. However, if something is wrong with the start-up or shut-down of an IPMI enabled IT device, we want to toggle the power during shut-down in the hope that a reset will solve the issue. Once we found the described issue with the simulator, we extended the model of the SU/SD controller with a storage of the start-up status of an IPMI enabled IT device.

In addition, the model triggered many discussions about the combined behaviour of the hardware and software involved in start-up and shut-down. This resulted in a clear description of responsibilities in the final concept. Also the exceptional system behaviour when errors occur has been elaborated much more compared to the traditional approach. Note that the modelling approach required a relatively small investment. The main POOSL model and the Java simulator were made in 40 hours; the tester and the stubs required another 10 hours.

The application of exhaustive model-checking techniques to the full model is not feasible, give the large number of concurrent processes and the use of queues for asynchronous communication. Scalability problems are, for instance, reported in [25], where a transformation of POOSL models to Uppaal [26], a model-checker for timed systems, is applied to an industrial application. However, it might be possible to apply these techniques to verify certain aspects on an abstraction of the model.

In the future, we want to use the test driver for the model to validate the behaviour of the SU/SD controller by means of model-based testing. Since the interface between the test driver and the model is equal to the interface between test driver and the real implementation, we might also use our test approach for the realized system when it become available. The idea is to use the test driver and a thin manually written adapter that makes an Ethernet connection between the test driver and the real implementation.

References

[1] S. R. Koo, H. S. Son, and P. H. Seong, "Nusee: Nuclear software engineering environment," in *Reliability and Risk Issues in Large Scale Safety-critical Digital Control Systems*, ser. Springer Series in Reliability Engineering. Springer London, 2009, pp. 121–135.

[2] B. Boehm and V. Basili, "Software defect reduction top 10 list," *IEEE Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[3] J. Westland, "The cost of errors in software development: evidence from industry," *The Journal of Systems and Software*, vol. 62, pp. 1–9, 2002.

[4] J. Groote, A. Osaiweran, M. Schuts, and J. Wesselius, "Investigating the effects of designing industrial control software using push and poll strategies," Eindhoven University of Technology, the Netherlands, Computer Science Report 11/16, 2011.

[5] J. Schmaltz and D. Borrione, "A functional approach to the formal specification of networks on chip," in *Formal Methods in Computer-Aided Design*, ser. LNCS, no. 3312. Springer–Verlag, 2004, pp. 52–66.

[6] M. Kaufmann, J. S. Moore, and P. Manolios, *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[7] S. M. Easterbrook, R. R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, "Experiences using lightweight formal methods for requirements modeling," *IEEE Trans. Software Eng.*, vol. 24, no. 1, pp. 4–14, 1998.

[8] S. Owre, J. Rushby, N. Shankar, and F. von Henke, "Formal verification for fault-tolerant architectures: prolegomena to the design of pvs," *Software Engineering, IEEE Transactions on*, vol. 21, no. 2, pp. 107–125, Feb 1995.

[9] A. Goodloe, C. A. Gunter, and M.-O. Stehr, "Formal prototyping in early stages of protocol design," in *Proc. of the 2005 Workshop on Issues in the Theory of Security*, ser. WITS '05. ACM, 2005, pp. 67–80.

[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada, "Maude: specification and programming in rewriting logic," *Theoretical Computer Science*, vol. 285, no. 2, pp. 187 – 243, 2002.

[11] MathWorks, "Matlab and Simulink," 2015. [Online]. Available: www.mathworks.com

[12] IBM, "Rational Rhapsody," 2015. [Online]. Available: www.ibm.com/software/products/en/ratirhapfami

[13] OMG, "Semantics of a foundational subset for executable UML models (fUML)," 2015. [Online]. Available: http://www.omg.org/spec/FUML/

[14] MagicDraw, "Cameo simulation toolkit," 2015. [Online]. Available: http://www.nomagic.com/products/magicdraw-addons/cameo-simulation-toolkit.html

[15] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *J. ACM*, vol. 31, pp. 560–599, 1984.

[16] R. Milner, *A Calculus of Communicating Systems*. Springer-Verlag, 1980.

[17] L. van Bokhoven, "Constructive tool design for formal languages; from semantics to executing models," Eindhoven University of Technology, the Netherlands," PhD thesis, 2004.

[18] SHE, "System-level design with the SHE methodology," 2015. [Online]. Available: www.es.ele.tue.nl/she/

[19] M. Geilen, "Formal techniques for verification of complex real-time systems," Eindhoven University of Technology, the Netherlands," PhD thesis, 2002.

[20] POOSL, "Parallel Object-Oriented Specification Language," 2015. [Online]. Available: poosl.esi.nl

[21] Intel, "Intelligent Platform Management Interface (IPMI) - specifications," 2015. [Online]. Available: www.intel.com/content/www/us/en/servers/ipmi/ipmi-specifications.html

[22] J. Groote, J. Keiren, A. Mathijssen, B. Ploeger, F. Stappers, C. Tankink, Y. Usenko, M. v. Weerdenburg, W. Wesselink, T. Willemse, and J. v. d. Wulp, "The mCRL2 toolset," in *Proceedings of the International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008)*, 2008.

[23] F. Systems, "Failures-Divergences Refinement (FDR)," 2015. [Online]. Available: www.fsel.com

[24] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2011: a toolbox for the construction and analysis of distributed processes," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 2, pp. 89–107, 2013.

[25] J. Xing, B. Theelen, R. Langerak, J. van de Pol, J. Tretmans, and J. Voeten, "From POOSL to UPPAAL: Transformation and quantitative analysis," in *10th Int. Conf. on Application of Concurrency to System Design (ACSD)*, 2010, pp. 47–56.

[26] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, ser. LNCS, no. 3185. Springer–Verlag, 2004, pp. 200–236.

On the Agile Development of Virtual Reality Systems

F. Mattioli[1], D. Caetano[1], A. Cardoso[1], and E. Lamounier[1]
[1]Faculty of Electrical Engineering, Federal University of Uberlândia, Uberlândia, Minas Gerais, Brazil

Abstract— *Processes for Agile software development present an iterative and incremental approach to computer systems, which focus on users' needs and embraces changes. Virtual Reality projects are strongly tied to rapid evolution of technology, and to the need for clients' feedback, during the whole project's life-cycle. In this work, a comparative evaluation of existing methodologies is presented and the application of agile software development methodologies in Virtual Reality projects is argued. Then, a proposal for an agile software development process for Virtual Reality systems is presented and its benefits are discussed.*

Keywords: Agile Development, Virtual Reality, Software Engineering

1. Introduction

Agile Software Development has, among its main features, an iterative and incremental approach to Software Engineering principles. This approach is suitable for Virtual Reality projects and offers, by its evolving nature, many benefits associated to risk management in software projects [1].

The word "agile" was first used in Software Engineering at 2001, by a consortium of software development methods specialists, who have written, at that time, the "Agile Manifesto" [2]. This manifesto highlighted some principles, shared by many different software development methods, which were thereafter called "Agile Methods" or "Agile Processes" [2], [3]:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Virtual Reality based systems require knowledge in different subjects, such as Computer Graphics, geometric modeling, multimodal interaction among others [4]. Some characteristics of these applications reveal the need for continuous improvement in their development process. Some of these characteristics can be highlighted:

- Rapid evolution of visualization and graphical processing technology [5].
- Customer's indecision and change of opinion, a critical concern when high-cost equipment is used [5].
- Need for implementation of prototypes, used to help customers in the solution's evaluation process.

Therefore, evolutionary development, adaptive planning and response to requirements' changes are major improvements to be considered on agile development of Virtual Reality systems.

2. Agile Software Development

Agile Software Development is an approach of software production focused on adaptability, which can be understood as the process' capability of responding to changes in markets, requirements, technology and development teams [6].

Sections 2.1 and 2.2 present a brief description of two agile methods: XP and Scrum.

2.1 Extreme Programming (XP)

Extreme Programming (XP) had its origins guided by the needs of small software development teams, working on projects with highly volatile requirements. XP is a light development method, which fundamentals include [7]:

- Unit tests are written before the code being tested. These tests are executed throughout the project life-cycle.
- Integration and testing are performed continuously, many times a day.
- The project begins with a simple architecture, that constantly evolves in an effort to increase flexibility and reduce unnecessary complexity.
- A minimal system is rapidly implemented and deployed. This minimal system will evolve according to project's directions.

Amongst the main benefits of Extreme Programming, the following are worth mentioning [7]:

- Do not force premature specialization of team members. All team members play different roles inside the development team, in a daily basis.
- Analysis and design are conducted throughout the whole project life-cycle, favoring adaptability and rapid response to project environment changes.
- Project infrastructure is built in an iterative way, following project's evolution and meeting its real needs.

Figure 1 presents the main elements of the XP process' life-cycle. User Stories are collected and used in requirements' specification and also in test scenarios definition. An Architectural Spike is conducted to elucidate the relevant solution elements, resulting on a System Metaphor.

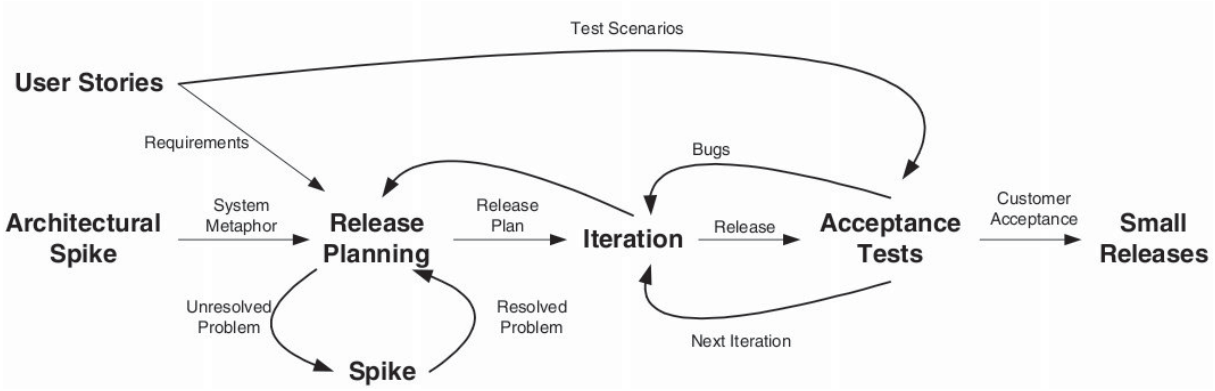


Fig. 1: XP life-cycle [8].

During Release Planning, architectural problems may arise. Each time an architectural problem is detected, a Spike is conducted to solve this problem. The resulting artifact is the Release Plan.

Each iteration targets a subset of functionalities. If a problem is detected during the iteration, Release Planning is carried again, and an updated Release Plan is written. At the end of the iteration, a release is made available and Acceptance Tests are conducted, using the test scenarios defined from the User Stories. If bugs are found during tests, another iteration is lead to fix them. If no bugs were found, the next iteration is started. When customer acceptance is confirmed, a Small Release (release of a working version of the software) is performed.

2.2 Scrum

Scrum is an empirical approach for managing software projects, based on the following principles: adaptability, flexibility, and productivity [9]. In Scrum, projects are divided into *sprints*. A *sprint* is a development iteration, with the typical duration of 30 days. Each *sprint* is associated to a set of tasks, whose priority is rather defined together with the clients. For each task, the remaining time to finish is estimated [10]. Tasks can be relocated, according to project's constraints.

In a nutshell, the Scrum process is composed by a set of rules, procedures and practices, favoring software development [8]. Figure 2 presents the Scrum process' life-cycle.

In the Scrum life-cycle, known requirements are grouped and prioritized in a product backlog [11]. A subset of these requirements, known as the "Sprint Backlog", contains the tasks assigned to a given *sprint*. From the "Sprint Backlog", tasks are elucidated in detail.

During the *sprint* - which is scaled for no more than 30 days - a daily review meeting is conducted. This daily meeting should not last long (15 minutes is a general suggestion), so that all project members can attend it [12]. In the daily meeting, team members are required to briefly answer three questions [13]:

1) What have I done since the last Daily Scrum?
2) What will I do between now and the next Daily Scrum?
3) What obstacles and roadblocks are in my way?

These answers have the objective of providing managers and developers with general information about the *sprint's* progress. Also, efforts can be grouped to help solving common problems, while experience can be shared in a daily basis.

Finally, at the end of each *sprint*, the new functionalities are demonstrated and tested, looking forward to stakeholders' approval.

2.3 XP and Scrum

Although complementary, XP and Scrum have different application, in different aspects of software development. While Scrum can be considered an agile project management tool, XP is more focused on the development side [14]. Scrum strengths include project's visibility in the market's context, continuous project management and improved collaboration between team members. XP motivation include a simplified requirements management approach and enhanced product quality. Both methodologies are based on iterative and incremental development [15].

Put together, Scrum and XP are valuable approaches, both on management and technical practices [14]. Therefore, in this work, an hybrid process is proposed. This hybrid process can benefit from Scrum management practices (such as "Sprint Backlog" and daily reviews), together with XP engineering practices (product quality, short iterations and test-driven development). This process is presented in detail on Section 4.

3. Virtual Reality Systems Development

The develpment of Virtual Reality Systems (VRS), as well as the development of any software, requires processes and development methods. In the particular case of VRS, methods and processes should be adequate to a rapidly

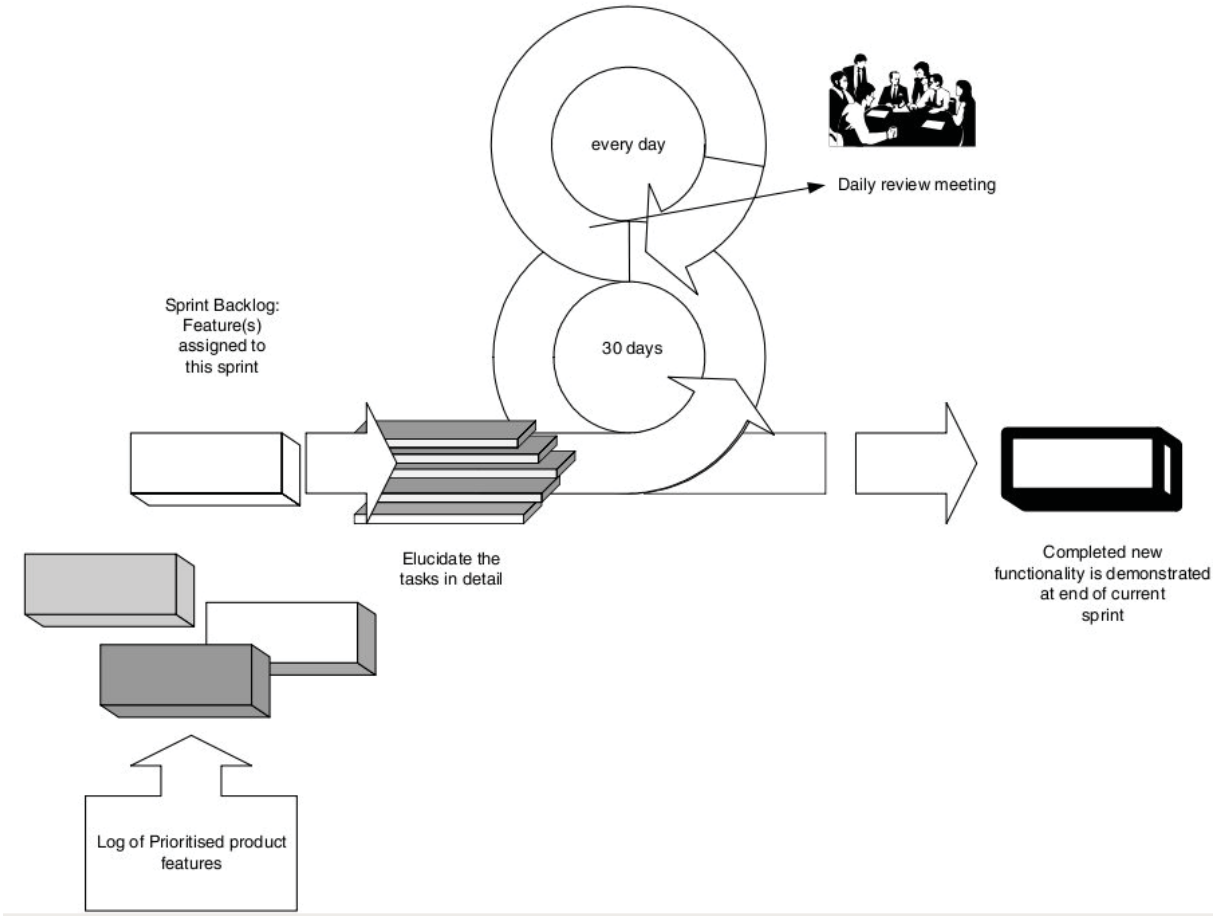


Fig. 2: Scrum life-cycle [8].

changing technological environment and to the particular aspects of user interaction.

Tori *et al.* present a development process that aggregates prototyping with iterative and evolutionary software development [5]. This process is based on Software Engineering models, adapted to the particularities of Virtual Reality Systems. The proposed process is composed of 5 stages, executed in each iteration: Requirement Analysis, Design, Implementation, Evaluation and Deployment. These stages are graphically represented in Figure 3.

Another development approach, also suitable for use on Virtual Reality Systems development is presented by Kim [16]. At first sight, this approach can be seen as an extension of the classic spiral model, adapted to Virtual Reality Systems characteristics, such as interaction models and scene modeling. Figure 4 presents the main elements of the proposed process.

Although both processes addressed in this section present strong influences from the structured approach, some VRS features are closely related to agile practices. Among them, one can highlight:

- The evolutionary nature of Virtual Reality Systems.
- The need for models that represent, iteratively, form, function and behavior of Virtual Reality Systems components.
- The better acceptance of systems which are developed with active participation of stakeholders, due to the constant need for evaluation and feedback.
- The need for exhaustive tests, aiming at reducing interaction problems between users and Virtual Reality Systems.

Based on these observations, the application of agile methods in Virtual Reality Systems development is discussed in Section 4.

4. Agile Development of Virtual Reality Systems

No software development process can guarantee, by itself, any improvement in productivity and reliability [17]. However, some characteristics are common in successful processes [18]:

- Iterative development: complex projects, with many modules, are more likely to face integration issues. An

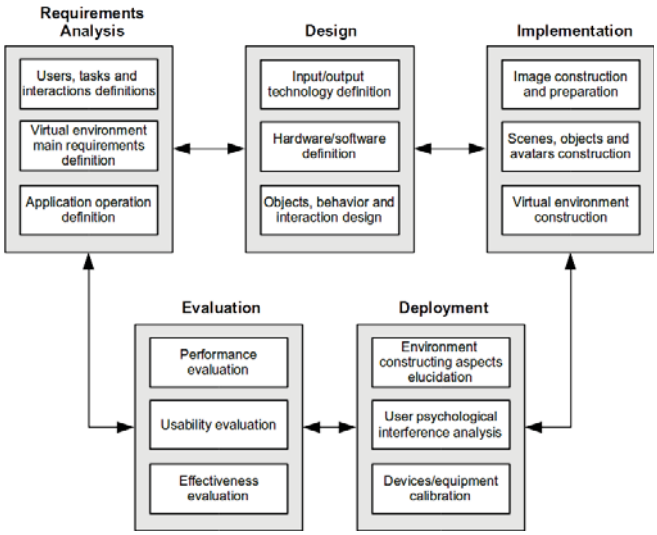


Fig. 3: VRS development process. Adapted from [5].

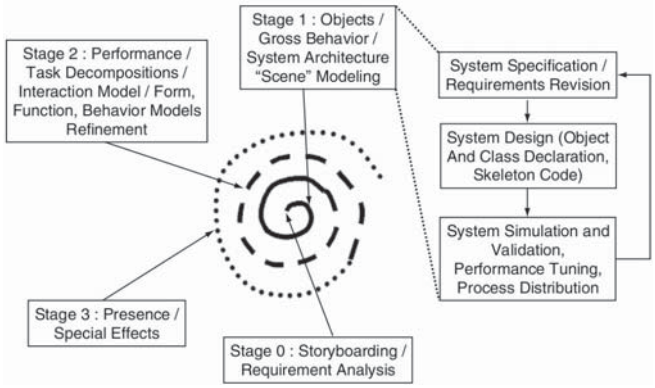


Fig. 4: VRS development process [16].

adequate iteration planning can reduce these integration issues and favor development process management.

- Process' continuous evaluation: even requirement-oriented development processes cannot make software projects totally immune to changes on development teams and on user requirements. The evaluation (and consequent adaptation) of the development process has a major importance throughout the project's life-cycle.
- Best practices: improvements associated to the use of development best practices [19], [20] and also design patterns [21] should be considered and discussed in software projects.

When adapting an existing process to a given context, the suggested approach is to customize the existing process, iteratively testing and refining this customization, in accordance with each project's characteristics [22]. During this customization, some principles might be observed [23]:

1) Larger teams require robust processes and methods.
2) Carried-over process complexity represent additional costs.
3) Critical applications require highly-detailed methods.
4) Clients' feedback and team communication reduce the need for intermediate documentation.
5) As the number of legal issues involved in a project increase, methods' level of detail should also increase.

From the presented literature review, this work's objective was defined: to propose a process model for the agile development of Virtual Reality systems. The proposed model - detailed in Section 5 - consists of a hybrid model, gathering elements from both XP and Scrum, adapted to the context of Virtual Reality systems development.

5. Results

In this section, a development process for Virtual Reality systems is proposed. The presented process is composed by 8 main activities: User stories / storyboards definition, architectural spike, interactivity requirements elucidation, iteration planning, spike, development, integration tests and client tests. Development is executed iteratively, and feedback received in past iterations is used to help planning the next ones.

By reviewing the state of the art of VRS development methods, some key features of these systems were defined:

- The evolutionary nature of VRS.
- Iterative building of high-fidelity models.
- The need for clients' feedback.
- The need for interaction and usability tests.
- The need for system modularization.

A VRS development process should keep these features in focus during the entire project life-cycle, in each of the activities presented above. In Figure 5, a graphical representation of the flow of activities in the proposed process is displayed. In the following sections, each of these activities is detailed.

5.1 User stories/storyboards

An user story is a brief description of a system functionality, from the user point of view. User stories are very helpful on requirement analysis because they provide developers with users' real expectations about the system.

When developing high-complexity graphical systems - such as VRS - text based user stories can be limited to detail users' needs. To overcome this, the proposed process suggests the use of storyboards, used to complement user stories. Storyboards are graphical sketches, elaborated by clients (or with their supervision), whose objective is to help developers on performing an accurate requirement analysis.

5.2 Interactivity requirements' analysis

Interactivity is the central aspect of many Virtual Reality systems, having a major role in these systems' usability. Thus, the detailed analysis and definition of interactivity

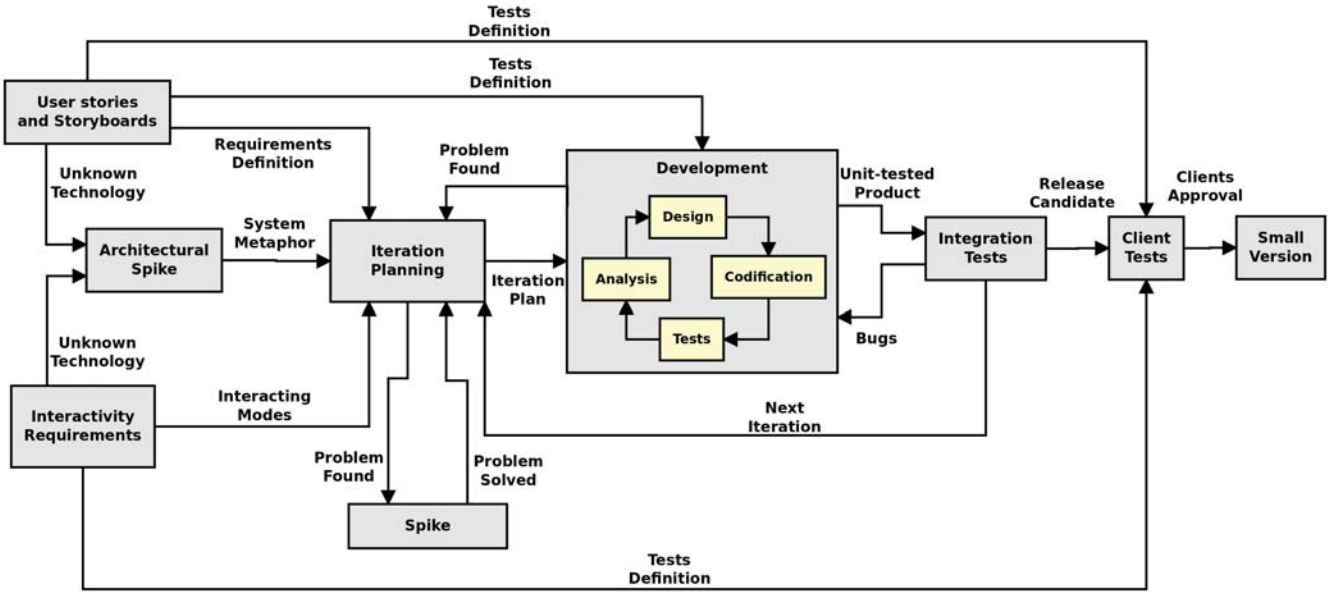


Fig. 5: Agile development process for VRS.

requirements has a prominent position ind the development process. In this activity, test applications - implementing the desired interaction methods - can be used. These applications should help developers and clients in the evaluation and viability analysis of the required interaction methods.

5.3 Architectural spike

In the architectural spike, a viability analysis of the new requirements is conducted. The requirements are extracted from user stories, storyboards and interactivity requirements' analysis. This activity has the goal of reducing risks related to unknown technology (for example, third-party libraries). The architectural spike results in the metaphor definition. This metaphor will be used by the development team to represent the subset of requirements in focus at the current iteration.

In the architectural spike, the resources available for requirements' implementation are investigated. This activity is very important in VRS development, since it suggests and encourages experimentation. Together with the "Interactivity requirements analysis", this activity is strongly related to technological advances in Virtual Reality.

5.4 Iteration planning

Iteration planning takes place at the beginning of each iteration. The resulting artifact - the iteration plan - addresses a subset of requirements, elucidated from the user stories, storyboards and interactivity requirements analysis. During iteration planning, each time a problem is detected, a spike is conducted, in order to investigate and propose possible solutions.

It's very important to highlight the adaptive behavior of iteration planning. Ideally, iteration planning should be flexible enough to embrace changes on application requirements and solutions to the problems found inside the iteration.

5.5 Spike

A spike is a small development cycle, whose main objective is to provide developers with possible solutions to a given problem. Inside the spike, test applications (or prototypes) can be built, to help developers on testing and discussing proposed solutions. If possible, clients' feedback can be used to guide the development team on the right direction, according to users' needs.

5.6 Development

Once the iteration plan is defined, development takes place. Development is composed by 4 main tasks, adapted from the consolidated Rational Unified Process [24]: analysis, design, codification and tests.

Analysis and design share the common goal of structuring the implementation of the iteration plan's requirements. A set of tests - proposed by the clients - is used to guide the development team on the implementation of the most important requirements, from the clients' point of view. Then, developers are requested to propose their own tests. With clients' and developers' tests defined, the codification task begins. Development activity is finished when the system successfully executes the proposed tests.

The evolutionary nature of the development activity should be highlighted. In the beginning, the system is composed by simplified models, that represent the main elements of the VRS. As the system evolves, these models are refined,

resulting in components whose shape and behavior are each time closer to the represented elements.

5.7 Integration tests

In each iteration, integration tests are conducted right after the development activity. The modifications performed in the current iteration will be integrated to the main system only after successfully passing these tests. If any problem is found during integration tests, development activity is restarted. Developers will then propose and test possible corrections to the problems found.

When the new version passes the integration tests, next iteration planning takes place. New requirements will be selected, according to the clients' defined priority. When a significant number of modifications is integrated to the main stream, a working version, called "release candidate", is produced and submitted to tests by the clients.

5.8 Client's tests

In this activity, clients are requested to perform functional, usability and interaction tests on the release candidate. If any problem is detected, or if any improvement is perceived by the client, user stories and interactivity requirements can be redefined. When clients' approval is obtained, a small version - which successfully implements a subset of proposed requirements - is delivered.

Specifically for the case of VRS development, interactivity tests play a major role in the overall development process. Therefore, interaction tests should be exhaustively executed by developers and clients, in order to avoid a significant drop in system's usability and efficiency, caused by poor interactivity.

6. Conclusions and future work

Agile software development processes and practices can be adapted to the development of Virtual Reality systems. In particular, the iterative nature, the embrace of requirement's changes and the importance given to tests are some of the characteristics that favors their application in VRS development, since these same characteristics are shared by many VRS projects.

Architectural spikes are considered a major improvement in the VRS development process, since they allow developers to conduct experiments in a constantly changing technology environment. The correct elucidation and definition of interactivity requirements has a strong effect on system's resulting usability and efficacy. Finally, given the subjectivity of some VRS concepts - such as systems' interactivity quality - stakeholders' participation in the development process leads to the production of improved quality systems, and results in well satisfied clients.

A quantitative evaluation of the presented process' application in a case study is an interesting proposal for future works. Also, the extension of the proposed process to other domains - such as Augmented Reality or mobile application development - is a valuable subject for future research.

Acknowledgments

This research is supported by FAPEMIG (Minas Gerais State Agency) to which the authors are deeply grateful, as well as to CAPES/Brazilian Ministry of Education & Culture and to the National Counsel of Technological and Scientific Development (CNPq).

References

[1] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, C. Jim, and H. K. A., *Object-oriented analysis and design with applications*, 3rd ed. Westford: John Wiley & Sons, 2007.

[2] K. Beck, M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. J. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," Available: http://www.agilemanifesto.org, 2001, accessed March 25, 2009.

[3] E. G. d. Costa-Filho, R. Penteado, J. C. A. Silva, and R. T. V. Braga, "Padrões e métodos ágeis: agilidade no processo de desenvolvimento de software [Agile patterns and methods: agility on software development process]," *5th Latin American Conference on Pattern Language of Programming*, vol. 5, pp. 156–169, 2005.

[4] G. J. Kim, K. C. Kang, H. Kim, and L. Jiyoung, "Software engineering of virtual worlds," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 131–138, 1998.

[5] R. Tori, C. Kirner, and R. Siscoutto, Eds., *Fundamentos e tecnologia de realidade virtual e aumentada [Virtual and augmented reality fundamentals and technology]*. Porto Alegre: SBC, 2006.

[6] A. Cockburn, *Agile software development*. Boston: Addison-Wesley, 2002.

[7] K. Beck, *Extreme programming explained: embrace change*, 2nd ed. Addison-Wesley Professional, 2004.

[8] J. Hunt, *Agile software construction*. London: Springer, 2006.

[9] M. A. Khan, A. Parveen, and M. Sadiq, "A method for the selection of software development life cycle models using analytic hierarchy process," in *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*. IEEE, 2014, pp. 534–540.

[10] V. Subramaniam and A. Hunt, *Practices of an agile developer*. Dallas: Pragmatic Bookshelf, 2006.

[11] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," in *Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on*. IEEE, 2014, pp. 189–196.

[12] M. Cohn, *Succeeding with Agile - Software development using Scrum*. Boston: Pearson, 2010.

[13] A. Stellman and J. Greene, *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. "O'Reilly Media, Inc.", 2014.

[14] M. Cohn, "Scrum & xp: Better together," Available: https://www.scrumalliance.org/community/spotlight/mike-cohn/april-2014/scrum-xp-better-together, 2014, accessed March 10, 2015.

[15] K. Waters, "Extreme programming versus scrum," Available: http://www.allaboutagile.com/extreme-programming-versus-scrum, 2008, accessed March 10, 2015.

[16] G. J. Kim, *Designing virtual reality systems: the structured approach*. London: Springer, 2005.

[17] F. Brooks, "No silver bullet: essence and accidents of software engineering," *IEEE computer*, vol. 20, no. 4, pp. 10–19, 1987.

[18] D. Pilone and R. Miles, *Head first software development*. Sebastopol: O'Reilly Media, 2008.

[19] B. W. Kernighan and R. Pike, *The practice of programming*. Reading: Addison-Wesley, 1999.

[20] A. Oram and G. Wilson, Eds., *Beautiful code*. Sebastopol: O'Reilly Media, 2007.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[22] J. Shore and S. Warden, *The art of agile development*. "O'Reilly Media, Inc.", 2007.

[23] A. Cockburn, *Agile software development: the cooperative game (agile software development series)*. Boston: Addison-Wesley Professional, 2006.

[24] P. Kruchten, *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

The impacts of absorptive capacity on improving software development: A preliminary study

Chung-Yang Chen[1], **and Jung-Chieh Lee**[1]
[1] Department of Information Management, National Central University, Jhongli City, Taiwan

Abstract - *Because information systems implementations depend on software development exercise, software process improvement (SPI) plays a critical role in the development of information systems. SPI is considered as an organizational learning process; it often needs external know-how and know-what that provide useful information and inspiration in implementation ad-hoc software processes. However, no research exists that focuses on a firm's ability to increase effective external SPI knowledge acquisition and utilization. Based on the dynamic capability theory, this paper attempts to explore how a firm's dynamic capability significantly affects SPI implementation success.*

The entire research consists of the exploration of related literature, the definition of and development of hypotheses and the research model, and the empirical investigation with new knowledge inquiry. This paper serves as a preliminary study of the research that focuses on the survey of relevant literature to support the research motivation and the model. Also, in this paper, a theoretical model draft and the hypotheses are developed to guide the following research of exploring and investigating the relationships between absorptive capacity and SPI success.

Keywords: Software process improvement (SPI); Capability maturity model integration (CMMI); Dynamic capability; Potential absorptive capacity (PAC); Realized absorptive capacity (RAC)

1 Introduction

Software process improvement (SPI) is particularly important for firms or business units as it enhances and sustains their competitive advantage in the business market. SPI is a complex and continually improving software processes program. Specifically, SPI is knowledge-intensive for firms, and the implementation of SPI often requires innovation and critical thinking to supplement original knowledge of software development with external knowledge. Therefore, SPI implementation often relies on SPI knowledge [4], skills, expertise, experiences, methodologies, and technical support from external sources (e.g. external mediating institutions such as SPI consulting firms and vendors, external knowledge bodies such as Capability Maturity Model Integration (CMMI), or the International Standards Organization (ISO)) to deal with challenges that arise during SPI implementation [6].

In both practice and theory, SPI is commonly recognized as an organizational learning process [4] because SPI implementation requires significant SPI knowledge and experiences from external sources, and employees must internalize these lessons [11]. During the learning process, the gap between the acquisition and the use of the acquired SPI knowledge within a firm is an important issue [18]. The gap exists when a company receives external knowledge without the capability of using it. However, the benefit may still exist in projects, since project members who have been educated with the new knowledge are able to tailor the organization's standard processes even though the new knowledge is not built in the standard processes. To achieve the expected SPI goals and receive the benefits due to the SPI implementation, the acquisition and utilization, be it at the project level or the organizational level, of SPI knowledge is required during SPI implementation. However, few SPI studies have focused on a firm's ability to acquire and utilize SPI knowledge to successfully implement SPI.

In literature, the concept of dynamic capability refers to a firm's latent abilities to renew and adapt its core competency over time [23, 26]. Further, Zahra and George [29] extend dynamic capability to include absorptive capacity (AC), which represents a firm's dynamic ability to acquire, assimilate, and apply knowledge from external sources. Moreover, scholars have noted that AC can be considered as a specific organizational learning process for the learning, implementing, and disseminating of external knowledge internally to strengthen, complement, or refocus on the knowledge mechanisms [9, 17, 22, 26]. In other words, AC assists firms in achieving positive outcomes, such as intra-organizational knowledge transfers [3], inter-organizational learning [28], and information technology (IT) and information systems (IS) implementation [2, 14, 19, 21].

To address a firm's ability to acquire and utilize SPI knowledge, this study focuses on the two categories of AC developed by Zahra and George [29], that is, potential absorptive capacity (PAC) and realized absorptive capacity (RAC), and investigates how these may influence SPI success. In this study, PAC refers to a firm's ability to identify, embrace, and assimilate external knowledge. On the other

hand, RAC represents a firm's ability to leverage newly absorbed knowledge and incorporate transformed knowledge into the development of innovation processes and operations [7]. Accordingly, insight into a firm's PAC and RAC is required for understanding how SPI knowledge acquisition and utilization affects SPI success.

Therefore, we are to explore how AC influences the success of SPI. Specifically, we address three research questions: (1) how PAC influences the success of SPI implementation within firms, (2) how RAC influences SPI success within firms, and (3) how PAC and RAC interrelate for SPI success. To answer these questions, this study proposes a research model that links PAC, RAC, and SPI success.

2 SPI Success

SPI helps firms integrate traditional organizational functions, and sets process improvement goals and priorities that update existing process systems to improve organizational performance [20]. SPI has played a critical role in helping firms achieve various business benefits. For example, SPI improves product quality, reduces the time to market, leads to better productivity, and reduces costs. To realize these benefits, the effective implementation of SPI requires effort and time, careful scheduling, resources, and suitable and useful knowledge [11, 13]. Decisions about SPI implementation are influenced by organizational factors, and several studies have analysed the critical success factors for SPI [13, 15].

In literature, Dyba [4] validated a theoretical model of SPI success factors and proposed an operational definition of the variables of SPI success. The study suggested that SPI success is defined by two indicators: improved organizational performance and the perceived level of SPI success, which includes cost reduction, decreased cycle time, and increased customer satisfaction. Dyba's theoretical model of SPI success factors has been applied in various studies. For example, Winter and Ronkko [28] investigated product usability metrics by adopting Dyba's SPI success factors. Egorova et al. [5] evaluated the effect of software engineering practices for industrial projects based on Dyba's work. In this study, we adopt Dyba's definition of SPI success as the dependent variable in the proposed model.

Prior studies have provided insight into identifying critical success factors for SPI. In spite of this, there is little or no research that focuses on how a firm's learning ability are placed to increase effective external SPI knowledge acquisition and acquired SPI knowledge utilization. To close this SPI knowledge gap, a firm's SPI knowledge activities in the context of SPI success should be investigated further. Therefore, this study adopts absorptive capacity (i.e. PAC and RAC) as the explanatory knowledge mechanism to investigate how organizational learning impacts SPI success.

3 Absorptive capacity

In literature, AC has played a critical role in ad-hoc investigation of IT and IS implementations [17]. According to the literature, AC was originally defined as a firm's ability to recognize the value of new, external information, assimilate it, and apply it to commercial ends [3]. AC also implies learning and acting in discovering scientific and technological activities outside the organization's limits [8]. It enables firms to gain superior organizational performance, innovation capability, and competitive advantage [9, 10].

Recently, Roberts et al. (2012) suggested several assumptions that underlie AC. First, AC depends on prior related knowledge. With some prior related knowledge, a firm can correctly select valuable and useful external knowledge. Second, an organization's AC depends on the AC of its individual members, which form a mosaic of individual capabilities. AC is firm-specific and embedded in the knowledge structures of a particular firm; hence, AC cannot easily be purchased. Third, accumulating AC is essential for efficiently utilizing the knowledge needed to face technological and market turbulence. In literature, AC is treated as a dynamic capability and a firm's AC affects its ability to reconfigure its existing substantive capabilities (Zahra and George, 2002; Jansen et al., 2005). According to the dynamic capability theory (Teece et al., 1997), Van den Bosch et al. [25] deemed AC as a high-level organizational ability. Zahra and George [29] further divided AC into PAC and RAC and distinguished the four dimensions of AC as being acquisition, assimilation, transformation, and exploitation.

Specifically, each of the dimensions is considered a capability that together produce AC, a dynamic capability of the organization [22], and these dimensions explain how AC influences a firm's knowledge mechanisms. Acquisition refers to a firm's ability to identify, acquire, and value external knowledge that is critical to operations. Assimilation refers to a firm's ability to analyse, process, interpret, and understand external knowledge. PAC enables a firm to be receptive to external knowledge and focus on the acquisition and assimilation of new external knowledge [29]. Transformation is a firm's ability to combine existing knowledge and the newly acquired and assimilated knowledge for future use. Exploitation refers to a firm's ability to integrate acquired, assimilated, and transformed knowledge into its operations to develop new processes, routines, operations, and systems. RAC enables a firm to transform and exploit the knowledge that has been absorbed [29].

As mentioned earlier, PAC and RAC can be performed as two separate, yet complementary, roles of AC in facilitating the use of new external knowledge [26, 29]. PAC is regarded critical because it enables a firm to make sense of and respond to its external business environments and challenges, enabling firms to adjust to change, explore new methods, and reshape

their knowledge base. Conversely, RAC is an essential foundation for forming and performing innovation [7]. From the organizational point of view, firms cannot exploit and apply knowledge without first acquiring it. Likewise, firms may have the capacity to acquire and assimilate external knowledge, but may not have the capacity to transform and exploit it into operations that enhance performance [29]. Besides the traditional concept that considering PAC and RAC as a whole, in this research, we are also to explore the possibility if PAC and RAC contributes to SPI success individually.

In literature, the effect of AC on organization's technological adoption and implementation has been an important subject. Many reports have demonstrated the importance of AC in IT and IS deployment and implementation. For example, Harrington and Guimaraes [8] indicated that AC establishes an external communication channel to gather useful knowledge that influences the implementation of new technologies. Srivardhana and Pawlowski [21] developed a theoretical framework to analyse AC that enables organizations to build new capabilities in creating and deploying enterprise resource planning (ERP) knowledge. Also, Saraf et al. [19] investigated the relationship between PAC and RAC in the assimilation of ERP systems and found that both PAC and RAC positively and directly impact ERP assimilation.

Although AC has been used in the IS/IT domain, it has not been addressed for the field of SPI, which is a critical and challenging task that enables the success of the aforementioned IT/IS development. Some previous studies have highlighted that the concept that SPI implementation depends on external prerequisites and essential SPI knowledge [12]. Therefore, this study considers that the role of AC is more pronounced when examining successful SPI implementation. Thus, we expect that AC as a firm's dynamic capability is relatively critical for SPI success. In the following section, we further review more literature to derive and develop the research hypotheses.

4 Hypotheses Development

Based on the dynamic capability theory [23] and considering the definition of AC, AC can facilitate organizations to focus on mechanisms of external knowledge. Vega-Jurado et al. [26] highlighted two important aspects that differentiate PAC and RAC. First, it is difficult to define a global measurement system because of the complex nature of AC and the organization's functional structure and arrangement. For example, PAC may be mastered by some educational units in an organization, while RAC is often performed by the (software) process group (EPG or SEPG) (SEI, 2010), since the group's job is to design and maintain organization's standard processes, in which new knowledge is applied. Based on such an organization's functional design, PAC and RAC may be performed and reviewed separately. Second, even though PAC and RAC are interrelated, their distinct effects should be examined separately. In other words, PAC emphasizes what we have learned; while RAC focuses on how the organization is improved by the learned knowledge. Therefore, our work focuses on the effects of PAC and RAC on SPI success, respectively. In the next paragraphs, the operational mechanisms of PAC and RAC are further elaborated respectively as follows.

In the context of SPI implementation, PAC begins with gathering the idiosyncratic SPI knowledge from external sources (i.e. SPI consulting firms and vendors). This facilitates identifying, acquiring, and evaluating external knowledge to determine what is compatible, suitable [29], and critical to the needs of SPI. Additionally, the question of how the acquired SPI knowledge is to coordinate with a firm's specific characteristics (e.g. technology, business strategy, people, and process) should be considered. In the next stage of PAC, assimilation enables a firm to analyse, interpret, and comprehend externally acquired SPI knowledge, and then disseminate the useful knowledge through the organisation. PAC exposes a firm to the external valuable SPI knowledge [26] used to stimulate improvement opportunities for software processes, routines, and operations. When learning from external sources, firms must be able to obtain and identify the external knowledge and translate it into 'local language'. PAC enables a firm to renew the knowledge base which is necessary for SPI, and increase the SPI knowledge acquisition required to implement improvements. Therefore, PAC likely leads to SPI success. Thus, we hypothesize that:

Hypothesis 1: PAC has a positive influence on SPI success.

In the entire learning operation of AC, RAC refers to the knowledge internalization by transforming and exploiting the acquired external SPI knowledge [29]. Transformation is regarded as the synthesis and integration of a firm's existing knowledge with the newly acquired and assimilated SPI knowledge. The internal SPI knowledge may be synthesized by the addition or elimination of knowledge, or by the conversion of external knowledge, with consideration of the firm's specific characteristics. Transformation also ensures that the synthesized knowledge is effectively and extensively transferred across the firm [8]. In the next stage, exploitation facilitates the transformed SPI knowledge to be incorporated into a firm's internal processes, operations, and routines for aligning and articulating the firm's SPI goals [1]. For example, a company may transfer the how-to-do from other company's implementation into its software process. Without RAC, organization may not receive a holistic benefit since the newly acquired knowledge is not able to be built into the company's processes and routines. Therefore, RAC seems critical to SPI success. Thus, we hypothesize that:

Hypothesis 2: RAC has a positive influence on SPI success.

Zahra and George [29] argued that external knowledge may not be transformed and exploited until it has been acquired and assimilated. Further, SPI is often aided by external knowledge [16]. As discussed, the development of AC might enhance SPI knowledge acquisition and utilization, and the implementation of SPI first requires effective acquisition of external knowledge (i.e. PAC). However, external knowledge cannot affect SPI success if the mechanism to transform and embed the absorbed knowledge into the firm's real processes, operations, and routines is not established (i.e. RAC). Thus, PAC is the first step to acquiring external SPI knowledge and RAC is the next logical step to exploit the new SPI knowledge. We therefore assume that RAC would mediate the relationship between PAC and SPI success. Thus, we hypothesize that:

Hypothesis 3: RAC mediates the relationship between PAC and SPI success.

Based on the aforementioned literature review and hypotheses, we are proposing a theoretical model that integrates PAC, RAC, and SPI success, as shown in Figure 1. Based upon this model foundation, we are then to further explore the variables that significantly impact the model to comprehend the entire model, as well as to conduct an empirical investigation to test the proposed research model and hypotheses accordingly.

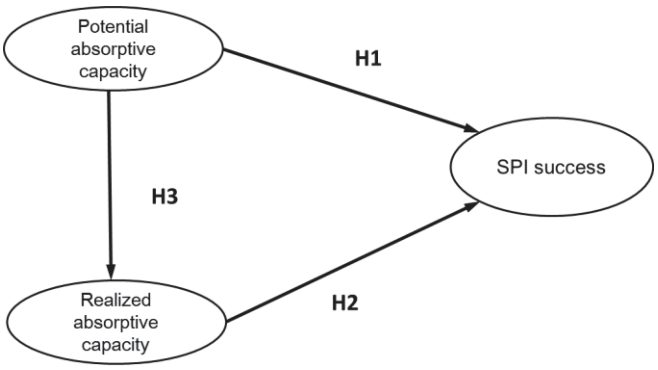


Figure 1. The proposed theoretical model.

5 Conclusions: what is next?

As the business environment becomes increasingly dynamic, many organizations have adopted SPI to achieve superior organizational performance. The pursuit of organizational performance relies on the organization's learning ability to acquire, process, and comprehend knowledge. In this paper, we have highlighted the importance of potential and realized absorptive capacity and have developed a draft of the theoretical research model to understand how organizational learning and its mechanisms, in terms of PAC and RAC, facilitate SPI success. It is hoped that through the discussion venue of this prestigious conference, valuable comments and suggestions can be obtained for helping the development of the model in the next research stage.

In the next stage of research, we are to complete the development of the proposed research model, and to test and verify the model. An empirical investigation will be conducted in the following study. We will further use a survey method for data collection in SPI-certified Taiwanese firms and examine the hypotheses using the statistical technique of partial least squares (PLS). PLS has commonly used in the IS literature. PLS is supposed to be distribution-free (i.e., the estimation is not affected by the complexity of the model, small sample size, or nonnormality of the data). Furthermore, PLS is also orthogonal and overcomes multicollinearlity problems [24].

Acknowledgement

This paper thanks the anonymous reviewers for providing useful comments and suggestions that help significantly improve the quality of the presentation. This paper also thanks research assistants, Mrs. Change, Mrs. Wang and Mr. Lee, for their help on collecting data and literature.

6 References

[1] Camisón, C. and Forés, B. (2010), "Knowledge absorptive capacity: new insights for its conceptualization and measurement", Journal of Business Research, Vol. 63, No, 7, pp. 707-715.

[2] Chen, J. S. and Ching, R. K. (2004), "An empirical study of the relationship of IT intensity and organizational absorptive capacity on CRM performance", Journal of Global Information Management, Vol, 12, No.1, pp. 1-17.

[3] Cohen, W. and Levinthal, D. (1990), "Absorptive capacity: a new perspective on learning and innovation", Administrative Science Quarterly, Vol. 35, No. 1, pp. 128–152.

[4] Dyba, T. (2005), "An empirical investigation of the key factors for success in software process improvement", IEEE Transactions on Software Engineering, Vol. 31, No. 5, pp. 410-424.

[5] Egorova, E., Torchiano, M. & Morisio, M. (2009). Evaluating the perceived effect of software engineering practices in the Italian industry. In: Trustworthy Software Development Processes, pp. 100-111, Springer Berlin Heidelberg.

[6] Feher, P. and Gabor, A. (2006), "The role of knowledge management supporters in software development companies",

Software Process: Improvement and Practice, Vol. 11, No. 3, pp. 251-260.

[7] Fosfuri, A. and Tribó, J. A. (2008), "Exploring the antecedents of potential absorptive capacity and its impact on innovation performance", Omega, Vol. 36, No. 2, pp. 173-187.

[8] Harrington, S. and Guimaraes, T. (2005), "Corporate culture, absorptive capacity and IT success", Information and Organization, Vol. 15, No. 1, pp. 39–63.

[9] Lane, P., Koka, B. and Pathak, S. (2006), "The reification of absorptive capacity: a critical review and rejuvenation of the construct", Academy of Management Review, Vol. 31, No. 4, pp. 833-863.

[10] Jansen, J. J., Van Den Bosch, F. A. and Volberda, H. W. (2005), "Managing potential and realized absorptive capacity: how do organizational antecedents matter", Academy of Management Journal, Vol. 48, No. 6, pp. 999-1015.

[11] Mathiassen, L. and Pourkomeylian, P. (2003), "Managing knowledge in a software organization", Journal of Knowledge Management, Vol. 7, No. 2, pp.63-80.

[12] Meehan, B. and Richardson, I. (2002), "Identification of software process knowledge management", Software process improvement and practice, Vol. 7, No. 2, pp. 47–55.

[13] Niazi, M., Wilson, D. and Zowghi, D. (2006), "Critical success factors for software process improvement implementation: an empirical study", Software Process Improvement and Practice, Vol. 11, No. 2, pp. 193–211.

[14] Park, J. H., Suh, H. J. and Yang, H. D. (2007), "Perceived absorptive capacity of individual users in performance of enterprise resource planning (ERP) usage: the case for Korean firms", Information & Management, Vol. 44, No. 3, pp. 300-312.

[15] Rainer, A. and Hall, T. (2002), "Key success factors for implementing software process improvement: a maturity-based analysis", Journal of Systems and Software, Vol. 62, No. 2, pp. 71–84.

[16] Ravichandran, T. and Rai, A. (2003), "Structural analysis of the impact of knowledge creation and knowledge embedding on software process capability", IEEE Transactions on Engineering Management, Vol. 50, No. 3, pp. 270-284.

[17] Roberts, N., Galluch, P., Dinger, M. and Grover, V. (2012), "Absorptive capacity and information systems research: review, synthesis, and directions for future research", MIS Quarterly, Vol. 36, No. 2, pp. 625-648.

[18] Rus, I. and Lindvall, M. (2002), "Knowledge management in software engineering", IEEE software, Vol. 19, No. 3, pp. 26–38.

[19] Saraf, N., Liang, H., Xue, Y. and Hu, Q. (2013), "How does organisational absorptive capacity matter in the assimilation of enterprise information systems", Information Systems Journal, Vol. 23, No. 3, pp. 245-267.

[20] Software Engineering Institute (SEI) (2010), "Capability Maturity Model Integration for Development", Carnegie Mellon University Press.

[21] Shih, C. and Huang, S. (2010), "Exploring the relationship between organizational culture and software process improvement deployment", Information & Management, Vol. 47, Nos 5-6, pp. 271-281.

[22] Srivardhana, T. and Pawlowski, S. (2007), "ERP systems as an enabler of sustained business process innovation: a knowledge-based view", Journal of Strategic Information Systems, Vol. 16, No. 1, pp. 51–69.

[23] Sun, P. Y. and Anderson, M. H. (2010), "An examination of the relationship between absorptive capacity and organizational learning, and a proposed integration", International Journal of Management Reviews, Vol. 12, No. 2, pp. 130-150.

[24] Teece, D., G. Pisano, A. and Shuen. 1997. "Dynamic capabilities and strategic management", Strategic Management Journal, Vol. 18, No. 7, pp. 509–533.

[25] Urbach, N., and Ahlemann, F. (2010). Structural equation modeling in information systems research using partial least squares. Journal of Information Technology Theory and Application, Vol. 11, No. 2, pp. 5-40.

[26] Van den Bosch, F., Volberda, H. and De Boer, M. (1999), "Coevolution of firm absorptive capacity and knowledge environment: organizational forms and combinative capabilities", Organization Science, Vol. 10, No. 5, pp. 551-568.

[27] Vega-Jurado, J., Gutiérrez-Gracia, A. and Fernández-de-Lucio, I. (2008), "Analyzing the determinants of firm's absorptive capacity: beyond R&D", R&D Management, Vol. 38, No. 4, pp. 392-405

[28] Volberda, H., Foss, N. and Lyles, M. (2010), "Absorbing the concept of absorptive capacity: how to realize its potential in the organization field", Organization Science, Vol. 21, No. 4, pp. 931–951.

[29] Winter, J. and Ronkko, K. (2010) SPI success factors within product usability evaluation. Journal of Systems and Software, Vol.83, No.11, pp. 2059-2072.

[30] Zahra, S. and George, G. (2002), "Absorptive capacity: a review, reconceptualization, and extension", Academy of Management Review, Vol. 27, No. 2, pp. 185–203.

An Infrastructure to Support Autonomic Control Loops in Dynamic Software Product Lines

Jane Dirce Alves Sandim Eleutério[1,2] **and Cecília Mary Fischer Rubira**[1]
[1]Institute of Computing, University of Campinas, Campinas, SP, Brazil
[2]Faculty of Computing, Federal University of Mato Grosso do Sul, Campo Grande, MS, Brazil

Abstract - *Dynamic Software Product Lines (DSPLs) use dynamic variability to adapt itself to the environment or requirements changes. The use of DSPLs is a way to achieve self-adaptive systems. Most existing DSPL solutions do not use autonomic control loop, although some solutions performed same loop activity implicitly. Moreover, there is the problem of how to identify the autonomic control loop or patterns present in DSPL solutions. Our approach is composed of three complementary parts: a product family of solutions for dependable DSPLs; a product line architecture for DSPLs, which explicitly use autonomic control loop patterns; and a model-driven infrastructure for DSPL to support dynamic applications. In this paper, we present a feature model of this proposed DSPL family and the preliminary results of the development of a product line architecture and our new infrastructure.*

Keywords: Self-Adaptive Systems; Dynamic Applications; Dynamic Software Product Line; Dynamic Composition.

1 Introduction

A Dynamic Software Product Line (DSPL) is a software product line that allows dynamic variability. Dynamic variability, also called late variability or runtime variability, can be represented using dynamic features, i.e., features that can be (de-)activated at runtime [1]. Also, context awareness, self-adaptation, and autonomous decision-making are some of the necessary properties for DSPL [2]. In particular, Self-adaptation can be implemented in several ways, allowing changes in the software structure to fix bugs, improve performance, increase availability and security, and change requirements [3]. Dynamically Adaptive Systems (DAS) should adapt their behavior or structure at runtime [4]. Dynamic Software Product Line could be classified as Dynamically Adaptive Systems [5]. In DAS, a system is usually composed of managed subsystem consists of application logic that provides the system domain functionality, and managing subsystem consists of adaptation logic that manages the managed subsystem [6]. The adaptation logic in self-adaptive systems typically involves the implementation of autonomic control loops, which defines how systems adapt their behavior to keep goals controlled, based on any regulatory control, disturbance rejection or optimization requirements [7]. The autonomic control loop has four components: Monitor, Analyze, Plan, and Execute, often defined as classic MAPE loop [6]. Consequently, the concept of self-managing from Dynamically Adaptive Systems is being increasingly used, combined with the increasing demand for more dependable systems [4]. Thus, this leads an advance on research related to Dependability and Fault Tolerance. Dependability of a system is the ability to avoid service failures that are more frequent or more severe than the acceptable [8]. Fault tolerance is a means to avoid service failures in the presence of faults during runtime [8].

According to a systematic mapping study about dependable DSPL [9], each approach often proposes a new framework or a new infrastructure, by suggesting new methodologies, commonly without follow a pattern or taxonomy. However, with all these different solutions, it is difficult to choose which, when and how to apply each solution in a particular case. A comparative study [10] analyzed the feasibility of achieving dynamic variability with DSPL-oriented approaches for the construction of self-adaptive systems. They used the following dimensions: "When to adapt" and "How to adapt". Moreover, the research on dynamic variability is still heavily based on the specification of decisions during design time. Besides, Bencomo *et al.* [10] ponder that many existing DSPLs are not as dynamic as researchers believe to be. Cheng *et al.* [11] reported the need to model explicitly the autonomic control loops as one of the major challenges. More specifically, autonomic control loops should be explicitly identified, recorded, and resolved during the development of self-adaptive systems, following autonomic control loop patterns, or self-adaptive reference architectures [11], [12]. However, most of the existing solutions related to DSPL do not apply autonomic control loops adequately [10], [11].

In this scenario, we detected the problem of how to identify the autonomic control loop pattern present in the proposed solutions for DSPL. As discussed in [11], [12], when the autonomic control loop is clearly identified, the maturity of the software engineering applied to self-adaptive systems increases. Moreover, it is also important to promote the separation of concerns between managed subsystems (application logic) and managing subsystems (adaptation logic). Therefore, our proposal describes a solution to support the creation of dynamic software product lines, including: (i) a family of solutions for dependable DSPLs; (ii) a self-adaptive architecture for DSPLs; and (iii) the

provision of a model-driven infrastructure for developing DSPL. This paper introduces the current state of our solution, presenting the preliminary results. Section 2 presents our proposal. We present the preliminary results of our research in Section 3. Section 4 presents related works. Finally, we conclude in Section 5.

2 The Proposed Solution

We propose the specification of a family of solutions for the development of dependable DSPL, using the studies found in the literature by means of a systematic mapping study about dependable DSPL [9]. We made an analysis of these solutions to identify the autonomic control loops and the adaptation patterns used. We also propose a definition of a self-adaptive architecture for DSPL. This self-adaptive architecture uses our created feature model for the derivation of DSPLs. We use the FArM method (Feature-Architecture Mapping) to map the feature model into a Product Line Architecture (PLA) [13]. Following this method [13], we refine iteratively the initial feature model into architectural components. However, we intend to provide to the software engineer a semi-automatic process to support the FArM method [13], by combining a set of process, models, tools, and source code generation. This semi-automatic process aims to support the creation of DSPLs by following the activities: (i) creation of the feature model with dynamic compositions, using a feature modelling tool; and (ii) generation of the product line architecture, using the FArM method [13] and the autonomic control loop patterns.

The generated product line architecture must have separation of concerns through the clear distinction between managing and managed subsystems. The managed subsystem has the components and/or services related to the application logic. The managing subsystem has the adaptation logic, including monitoring, analysis, planning and implementation components for adaptation, according to the autonomic control loop patterns. For the generation of self-adaptive architecture, we use the autonomic control loop patterns presented by Weyns *et al.* [6]: Coordinated Control, Information Sharing, Master/Slave, Regional Planning, and Hierarchical Control.

We also propose a definition and implementation of a model-driven infrastructure for DSPLs instantiation. Our infrastructure uses the feature model and the product line architecture to derivate family members, which are DSPLs. Our infrastructure is composed of a dynamic component framework with a reflective architecture. The managed subsystem (application logic) is oblivious to the managing subsystem (adaptation logic). At runtime, the managing subsystem intercepts the running system when an adaptation is required. Managing subsystem is organized meeting the MAPE autonomic control loop, which is divided into Monitor, Analyze, Plan, and Execute [6]. Besides, there is a knowledge base that supports the required information flow throughout the loop. Therefore, this proposal specifies a product line of DSPLs, where each member of the family would not be a finalized DSPL, but a framework for the creation of DSPL according to the chosen configuration. It is possible to use this derived framework for the creation of a DSPL, and it must have only the chosen features at derivation time.

3 Preliminary Results

3.1 Feature Model for Dependable DSPLs

We performed the product family modeling of solutions for dependable DSPLs, using as inputs the related solutions presented in [9]. We modeled variability and commonalities of the family of dependable DSPLs in a differentiated way to represent the autonomic control loop activities. Each phase, for example, *Monitoring*, was modeled as subdivided features that will compose its essence. As a result, Fig. 1 shows the feature model of the DSPL family. This feature model of dependable DSPL family provides three major variation points: (i) the selection of MAPE pattern - centralized or decentralized. Whether decentralized, the designer can choose one of the five decentralized MAPE patterns; (ii) the selection of specificity for each MAPE activity; and (iii) the choice of sensors - the designer will be able to choose the monitoring sensors to be used.

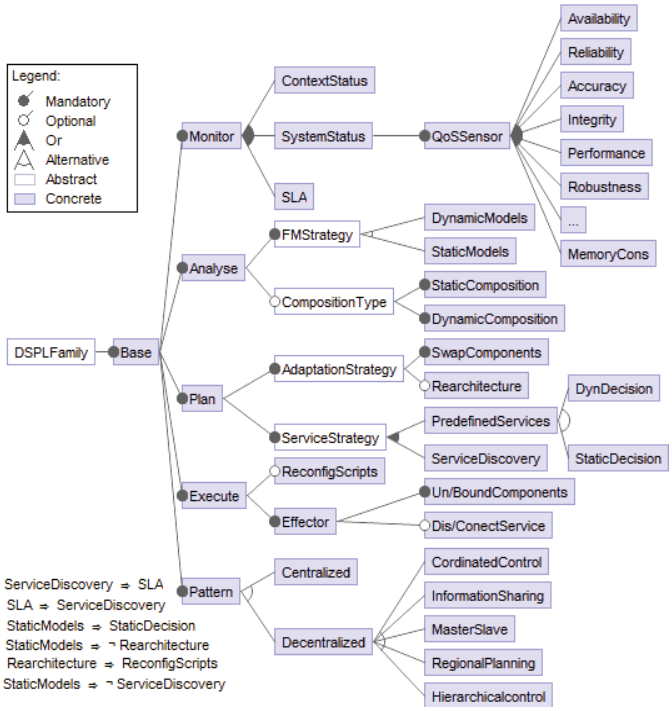


Fig. 1. The feature model of a family of solutions for dependable DSPLs.

3.2 Feature Model with Dynamic Features

As we had no tool available to perform the modeling of dynamic features, we decided to modify the FeatureIDE plug-in for Eclipse [14] to meet our needs. The feature

model may contain static and dynamic compositions. On static composition, decision-making is taken at design time, unlike on dynamic composition, which occurs at runtime. A dynamic composition is a relationship between dynamic features. Therefore, the system can compose and activate dependent features only at runtime [15]. With the new plug-in version, it is possible to specify the feature model, by defining some features as dynamic (green dashed border in Fig. 2). Therefore, the existing variation points in this feature become a dynamic composition. Fig. 2 shows the dynamic feature *VisaPaymentService*, which its associated variation points are a dynamic composition, i.e., a choice between *Visa1*, *Visa2*, and *Visa3*.

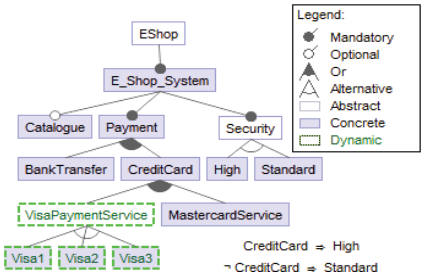


Fig. 2. An example of a feature model with dynamic composition.

3.3 Architecture and Infrastructure

With our model-driven infrastructure, the software engineer can choose what monitoring sensors will be applied to each dynamic feature. Thus, these sensors will be applied to their implemented components and/or services. From the model feature created by the engineer using our plug-in, an XML file is generated. With the created XML file, our infrastructure creates an XML configuration file for mapping features and components and/or services, that will be implemented by the programmer. With both XML files and selected MAPE pattern, our infrastructure generates a preliminary version of the self-adaptive product line architecture (PLA). These steps compose a semi-automatic process to support the FArM method [13], which was proposed to provide to the software engineer a means to create DSPLs. This semi-automatic process encompasses a set of process, models, tools, and source code generation. Fig. 3 represents this process. The generated self-adaptive product line architecture consists of: (i) components of managed subsystem (application components); (ii) MAPE components of managing subsystem, according to the selected pattern; and (iii) monitoring sensor components applied to managed subsystem components. We plan to generate graphically this architecture to enable the designer to perform the necessary changes. Fig. 4 shows a representation of this architecture applied to the example of *VisaPaymentService* (Fig. 2). Also, the generated infrastructure is a framework, which managing subsystem components will be fulfilled with source code. Besides, the managed subsystem components are partially implemented. In other words, the created infrastructure will be composed of the full source code required by the managing subsystem and the structure (packages and files) and partial source code required by the managed subsystem. Our model-driven infrastructure encompasses the semi-automatic process, the feature model of DSPL family (Fig. 1), the self-adaptive architecture, and the tools used to model and to generate the required source-codes by the subsystems.

Presently, we are ending the coding phase of our infrastructure. We are implementing Sensors and Effectors using APIs for introspection and reconfiguration provided by the OSGi platform. We are implementing software components according to COSMOS*. At runtime, to bind software components in the managed subsystem, we employ a service locator, instead of traditional architectural connectors. The 'service locator' is used by Effectors and acts as a simple runtime linker.

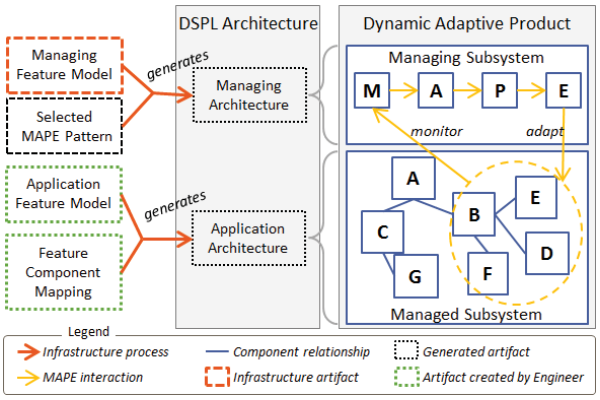


Fig. 3. Our semi-automatic process.

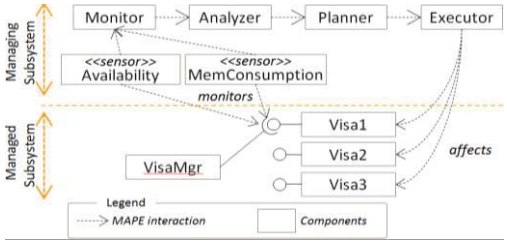


Fig. 4. An example of a generated Self-adaptive architecture.

4 Related Work

A systematic mapping study of dependable dynamic software product line showed us the main contributions and limitations of these solutions [9]. Batory *et al.* [16] proposed a decomposition of framework structures and instances in primitive and reusable components, reducing the source-code replication and creating a component-based product line in the context of object-oriented frameworks. Camargo and Masiero [17] proposed an aspect-oriented framework family, called Crosscutting Framework Family, where during the development of software, only the resources required by the software are used, resulting in a well-structured design, providing higher levels of maintainability and reusability. Oliveira *et al.* [18] extended the Camargo and Masiero approach [17] and introduced the concept of Framework Product Lines (FPL), where each family member is a framework, applying the concept of

frameworks in SPL. The approaches described in [16]–[18] differ from our proposed research. Our proposed approach is a family of dynamic software product lines, i.e., the concept of software product line applied to another (dynamic) software product line. This concept is not explored by previous studies, making our proposal an unique contribution in this research field.

5 Conclusions

In this paper, we presented our research proposal, which describes a solution that aims to help the creation of self-adaptive systems using DSPL techniques. Three complementary parts compose our approach. First, we proposed a family of solutions for dependable DSPLs, using as inputs previously identified solutions by means of systematic mapping study [9]. We concluded this activity and presented its feature model in Fig. 1. Second, we proposed a definition of a Product Line Architecture (PLA) for DSPL, which uses our created feature model for DSPL derivation. We use a semi-automatic process to support the creation of DSPLs according to the following activities: (i) modeling of the feature model with dynamic compositions, using a feature modelling tool; and (ii) generation of product line architecture, using the FArM method [13] and autonomic control loop patterns. Third, we proposed a model-driven infrastructure for instantiation of DSPL, aiming to support the creation of self-adaptive systems. Our primary objective is to build an infrastructure that will provide several approaches to implementing DSPLs in a single solution. Our infrastructure will meet the autonomic control loop patterns, defined by Weyns *et al.* [6]. This activity is still under development, but we presented some preliminary results in the Section 3. Currently, we are ending the coding phase of our proposed infrastructure.

Despite our research is ongoing, we identified some contributions. The modeling of the solutions family of DSPLs helps us in a better understanding of how the solutions were implemented and what are their advantages and limitations, helping on the future development of new solutions. Our infrastructure provides focus on MAPE patterns and highlights the autonomic control loops, meeting a part of the challenges listed in [10]–[12]. Otherwise, our proposal focuses on non-functional variability to explore the actual software variability in the development of DSPL and its techniques. When ready, our infrastructure will allow the software engineer to select the most suitable DSPL techniques at design time in accordance with its domain, project needs, or requirements. The infrastructure will automatically generate the major parts of the source code required by the family of dynamic applications.

6 Acknowledgement

The authors thank to the Fundect-MS and Unicamp for financial and logistic support.

7 References

[1] J. van Gurp, "Variability in Software Systems The Key to Software Reuse," Blekinge Institute of Technology, 2000.

[2] S. Hallsteinsen et al., "Dynamic Software Product Lines," Computer, vol. 41, no. 4, pp. 93–95, 2008.

[3] P. K. McKinley et al., "Composing adaptive software," Computer, vol. 37, no. 7, pp. 56–64, 2004.

[4] H. J. Goldsby et al., "Digitally Evolving Models for Dynamically Adaptive Systems," in ICSE Workshops SEAMS '07., 2007, p. 13.

[5] N. Bencomo et al., "Dynamically Adaptive Systems are Product Lines too : Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems," in 2nd DSPL workshop (SPLC 2008, Volume 2), 2008, pp. 117–126.

[6] D. Weyns et al., "On patterns for decentralized control in self-adaptive systems," Lect. Notes Comput. Sci., vol. 7475 LNCS, pp. 76–107, 2013.

[7] H. A. Müller et al., "Autonomic Computing Now You See It, Now You Don't — Design and Evolution of Autonomic Software Systems," Softw. Eng., vol. 5413 LNCS, pp. 32–54, 2009.

[8] A. Avizienis et al., "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable Secur. Comput., vol. 1, no. 1, pp. 11–33, Jan. 2004.

[9] J. D. A. S. Eleutério et al., "Dependable Dynamic Software Product Line – a Systematic Literature Review," Institute of Computing - UNICAMP, IC-15-03, 2015.

[10] N. Bencomo and J. Lee, "How dynamic is your Dynamic Software Product Line?," Softw. Prod. Lines Going Beyond, no. 6287 LNCS, 2010.

[11] B. H. C. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," Softw. Eng. Self-Adaptive Syst., vol. 5525 LNCS, pp. 1–26, 2009.

[12] Y. Brun et al., "Engineering Self-Adaptive Systems through Feedback Loops," Softw. Eng. Self-Adaptive Syst., vol. 5525 LNCS, pp. 48–70, 2009.

[13] P. Sochos et al., The feature-architecture mapping (FArM) method for feature-oriented development of software product lines. IEEE, 2006.

[14] T. Thüm et al., "FeatureIDE: An extensible framework for feature-oriented software development," Sci. Comput. Program., vol. 79, pp. 70–85, 2014.

[15] J. Lee et al., "Engineering Service-Based Dynamic Software Product Lines," Computer, vol. 45, no. 10, pp. 49–55, Oct. 2012.

[16] D. Batory et al., "Object-Oriented Frameworks and Product Lines," Softw. Prod. Lines - Exp. Res. Dir., vol. 576, pp. 227–247, 2000.

[17] V. V. Camargo and P. C. Masiero, "An approach to design crosscutting framework families," in Proc. of the 2008 AOSD, 2008, pp. 1–6.

[18] A. L. de Oliveira et al., "Investigating framework product lines," in Proc. of SAC '12, 2012, p. 1177.

Psychological Considerations for Agile Development Teams

James A. Crowder
Raytheon Intelligence, Information and Services, Aurora, CO, USA

Abstract— *For modern managers, one has to adopt a new philosophy, or psychology for dealing with agile development teams. While process is important to ensure the team delivers quality software that meets customer requirements, it is important to understand that the Agile Method is geared around more of an informal approach to management, while putting more time, effort, and emphasis on flexibility, communication, and transparency between team members and between the team and management. It promotes an environment of less control by managers and more facilitation by managers. The role of the manager takes on a new psychological role, one of removing roadblocks, encouraging openness and communication, keeping track of the change-driven environment to ensure that the overall product meets in goals and requirements, while not putting too much control on the ebb-and-flow of the agile development process. Change is no longer wrong, the lack of ability to change is now wrong. Here we discuss the new "soft" people skills required for modern managers, and how they add/detract from modern agile development. How to recognize the skills, how to utilize the skills, and how to build teams with the right "mix" of personalities and soft people skills for effective and efficient development efforts [1].*

Keywords—Agile Development, Agile Team Building, Agile Management

1. PEOPLE, NOT PROCESSES AND TOOLS

Companies have spent decades designing, creating, implementing, and executing tools required to bid and manage development projects. Once major category of tools is prediction tools like CiteSeer© and COCOMO© (Constructive Cost Model) have been used since the late 1990's to provide "objective" cost bids for software development. A later version of COCOMO, COSYSMO© (Constructive Systems Engineering Model) attempts to provide objective systems engineering bids also. All of them are based on the antiquated notion of Software Lines of Code (SLOC). Productivity metrics are all based on the lines of code written/unit time. They try to estimate the life-cycle cost of software, including designing, coding, testing, bug-fixes, and maintenance of the software. But ultimately it comes down to Software Lines of Code/Month (SLOC/Month). While many will claim these are objective tools for helping to determine the staff loading necessary for a software/systems development project. In each tool there are dozens of parameters which are input by the operator, each of which has an effect on the outcome of the cost model. Parameters like efficiency (average SLOC/Month), familiarity with the software language used, average experience level, etc. can be manipulated, and usually are, to arrive at the answer that was determined before the prediction tool was used [2].

Many other tools are utilized to measure the performance (cost and schedule) of projects once they are in execution. These measurement tools measure how the project is progressing against its pre-established cost and schedule profile, determined in the planning phase of the program/project. What none of these tools, cost estimation, performance metrics tools, etc. take into account are the actual agile team and their dynamics. The makeup of the each agile team and the facilitation of each team is as important, if not more important, than the initial planning of the project. If the Agile Manager/Leader is not cognizant of the skills necessary not to just write code, but to work cohesively as an agile team, then success is as random as how the teams were chosen (usually by who is available at the time). Grabbing the available software engineers, throwing them randomly into teams, and sending them off to do good agile things will usually result in abject failure of the project; or at least seriously reduced efficiency. This may sound like an extreme example, but you would be surprised how many agile development projects are staffed in just this fashion. Many managers point to the following graph (Figure 1) as the reasons not to go to the expense of changing all their processes to accommodate agile development.

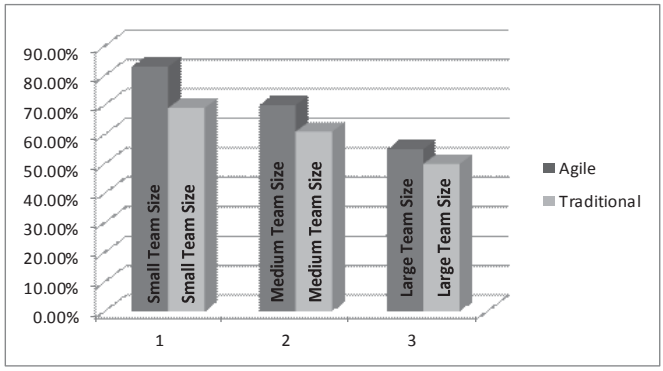


Figure 1 – Efficiencies between Traditional and Agile Development

While in each category agile development produces a higher efficiency than traditional software development methods, the increase is not as dramatic as the promises made by agile advocates and zealots. Classical managers find this graph disturbing and feel smugly justified in their classical software development/execution/control methods. This is especially true for large teams. The data for this graph was taken from 50 of each size project, both agile and traditional. What are not illustrated by this graph are the management methods

utilized across the traditional vs. agile programs/projects; the team make-up, how the teams were chosen, or any discussion of the types of issues that were encountered during the development process. And while it's clear that under any team size agile development has increased efficiency over traditional methods, and, as expected, smaller team sizes produce better results with agile methods, understanding the true nature of the agile team process and applying the psychology of agile management can achieve even greater efficiencies.

Placing the emphasis on the individuals in the agile development teams rather than on process or tools means understanding people, recognizing their strengths (not only in terms of programming skills, but also in terms of soft people skills), understanding the differences between people of different backgrounds and how the differences affect team dynamics. This is the first generation where it is possible to have 60 year-old software engineers in the same agile development teams with software engineers in their early 20s. The generational differences in perspectives can severely hamper team dynamics, and therefore team efficiencies will suffer greatly if they are not dealt with appropriately and the team members are not trained in how to function in an agile development team. All members of the teams need to be able to understand and come to grips with four main components of agile development, illustrated below in Figure 2. While there are other components that are important, without a good handle and agreement on these, agile development teams are in trouble from the start [3].

Figure 2 – Four Main Components of the Agile Development Process

As explained, Figure 2 represents four of the major components of the Agile Development Process that must be embraced by the agile development team in order to have a successful and efficient development process. As important are the skills, or philosophies, that the manager of the program/project must embrace and practice in order for the teams to be able to function in an agile environment and have the best chance for success. Figure 1-4 provided a high-level look at the skills of the effective agile manager/leader. The descriptions of these skills are:

1. **Effective Communicator**: The effective communicator fosters and increases trust, is transparent, considers cultural differences, is able to be flexible in delivery of communications, encourages autonomy, role models, exudes confidence to solve problems, handle whatever comes up and has the courage to admit when they are not sure and willingness to find out. They are willing to work side by side verses competitive with followers. They have the ability to communicate clear professional identity and integrity, their values are clear and so are their expectations. The effective communicator communicates congruence with values and goals, as well as being a role model of ethical and culturally sensitive behavior and values.
2. **Diplomat**: The diplomat considers the impacts on all stakeholders and how to follow up with all those effected, even if it is delegated. There is willingness to consult cultural experts.
3. **Effective Listener**: The effective listener checks that they understand the meaning being portrayed, and goes with an idea even if they disagree until the whole idea is expressed and the originator can think through the complete thoughts with the leader.
4. **Analytical Thinker**: The analytical thinker must be able to see the forest and the trees. The analytical thinking manager/leader must be able anticipate outcomes and problems, and explore how they might anticipate handling them, walking through possible solutions. They must initiate Professional Development of team members. They think about the how, not just the what-ifs.

2. Establishing Agile Team Goals

For the effective agile project/program manager, it is crucial early on to establish goals and objectives that establish the atmosphere for each sprint development team. Understanding how much independence each developer is allowed, how much interdependence each team member and each team should expect, and creating an environment that supports the agile development style will provide your teams with the best chance for success. Below is a list of agile team characteristics and constraints that much be defined in order for the teams to establish a business or development "rhythm" throughout the agile development cycle for the program/project. Each will be explained in detail in its own section, but general definitions are given below:

1. **Define and Create Independence**: Independence is something many developers crave. In order for agile development to be successful, there must be a large degree of independence and need to feel an atmosphere of empowerment; where the developers are free to create and code the capabilities laid out during the planning phase of each sprint. This requires a level of trust. Trust that the developers and the leader all have stakeholders in

mind. Trust that the developer is working toward the end product [4]. Empowerment at the organizational level provides structure and clear expectations [5]. At the individual level allows for creativity. Independence means having a voice and yet operating under company structure of policies and procedures. Independence is also a sense of knowing that the developer is good and what they do. There is no need to check in too frequently with the leader, but enough to keep the teamwork cohesive.

2. **Define and Create Interdependence**: While independence is a desired and necessary atmosphere for agile teams, the agile manager must also establish the boundaries where individual developers, and development teams, must be interdependent on each other, given that the goal is create an integrated, whole system, not just independent parts. Interdependence is being able to rely on team members [6]. The end goal will require a level of commitment from each person with a common mission in mind. The trust that all individuals on the team have all stakeholders in mind. This gets the whole team to the common goal and reduces each member motivated solely for their own end goal.
3. **Establish Overall, Individual, and Team Goals and Objectives**: setting the project/program overall goals, team goals, and individual goals and objectives up front, and at the beginning of each sprint help each team and individual team member to work success at all levels of the program/project. This can help to identify strengths of individuals so that the team can use its assets to their highest production. This also allows room for individual development and growth along with a place for passions. This also sets up clear expectations, say of the overall and team goals. There may be some individual development that is between the leader and the developer that stays between them. This would also build individual trust between members of the team and between the leader and the developers.
4. **Establish Self-Organization Concepts**: self-organizing teams is one of the holy grails of agile development teams. However, self-organization is sometimes a myth, mostly because teams are not trained into how to self-organize. People do not just inherently self-organize well. If not trained, the stronger personalities will always run the teams, whether they are the best candidates or not [7]. Self-organization can be nearly impossible when there are very structured people coupled with not-so-structured people. There may be some work that the leader can do to promote self-organization. Part of that is opening communication, building dyads, calling behavior what it is, and being transparent so that others will follow. It may be helpful for team members to get to know strengths of other members and how each member can be helpful to each individual.
5. **Establish Feedback and Collaboration Timelines and Objectives**: given the loose structure and nature of agile development, feedback early and often is crucial to allowing the teams to adapt to changing requirements or development environments. Also, customer collaboration and feedback at each level in the development allows the teams to adjust and vector their development efforts, requirements, etc., to match customer expectations at all points in the development cycle. Feedback timelines can increase trust and clarify all expectations. It is nice to know when you need to change a direction, when you need to change it, instead of later when you had already put so much work into the project. The more feedback is modeled and practice the more natural it becomes and becomes more automatic. This builds on the independence and interdependence of the team and individual stakeholders.
6. **Establish Stable Sprint Team Membership**: choosing the right teams is important for success in an agile development program/project. Creating teams that are not volatile (changing members often) is essential to continued success across multiple sprints. If the teams constantly have to integrate new members, efficiency will suffer greatly. New expectations and explanations will take up much time that could be used for developing. A trusting team can be an efficient team. The more often it changes the more work needs to be done to build the trust. There may be increased commitment from those that work on a cohesive team with high trust levels and knowledge of on another [3].
7. **Establish Team's Ability to Challenge and Question Sprints**: if the teams are going to be allowed individual and team empowerment, then they must be allowed to challenge and question sprint capabilities and content across the development cycle. Forcing solutions on the teams fosters resentment and a lack of commitment to the program/project. If you've built the right team, you should listen to them. It seems more productive to work on something that makes sense to you, instead of handed down by others. The ability to challenge and question will lead to better understanding and more commitment to the end goal.
8. **Establish an Environment of Mentoring, Learning, and Creativity**: invariably, teams are composed of a combination of experience levels. This provides an excellent atmosphere of mentoring and learning, if the agile manager allows this. This must be built into the sprint schedules, understanding that an atmosphere of mentoring, learning, and creativity will increase efficiencies as the team progresses; not just on this project, but on future projects as well, as the team members learn from each other. Keep in mind that experienced developers can learn from junior developer too, as the more junior developer may have learned techniques and skills that were not previously available to more senior developers. The learning environment promotes growth. An environment that fosters learning decreases negative feelings of one's self, and thus other people. An environment that fosters learning isn't run by guilt, or feelings of not being good enough, or doing something wrong. A learning environment allows people to grow and

the mentor helps the individuals self-determine the direction they want to develop. The learning environment will foster older members learning from younger members as well. People will want to learn more and more and reduce competitiveness that can destroy a team. The competitiveness can come out as a good product not team dynamics. Transparency can help individuals feel more comfortable with learning. This can show that is ok to have areas of development and that everyone has room to grow.

9. **Keep Mission Vision always out in Front of Teams**: many believe that an established architecture is not required for agile development. This is absolutely wrong; a solid architecture is even more important during agile development, so each team and team member understands the end goals for the system. However, in order to for the architecture and software to stay in sync, the systems engineering must also be agile enough to change as the system is redesigned (or adapted) over time [8]. Agility is not free from structure but the ability to move about within the structure.

3. Independence and Interdependence: Empowerment

Locus of Empowerment has been conceptualized as a function of informed choice and self-determination and has been linked to the concepts of self-efficacy and locus of control as it applies to agile team membership [9]. Self-understanding and empowerment, in relation to development opportunities and factual strength/weakness assessment, represents an important underlying component of feelings of self-empowerment within an agile development team [10]. Locus of empowerment, and its counterpart, Locus of Control, help to establish both independence and interdependence for agile team members. Determining those things each team member is "empowered" to make decision on and work independently provides each person with a sense of autonomy, allowing them to work at their peak efficiency without interference or too much oversight control over their work. Establishing the Interdependence, or those things which are outside of the control of the team member, defines communication lines and those things which are necessary to collaborate on, or get inputs from other team members to facilitate integration and validation of "system-wide" capabilities [11]. What follows is a discussion of Locus of Empowerment.

The notion of Locus of Empowerment is an interactive process that involves an individual team members' interaction with the team and the manager [12], allowing each team member to develop a sense of acceptance into the team, develop a sense of where they belong in the team, self-assessment of skills, and determination of their self-efficacy; their ability to function and participate both on an individual level, and as part of an agile development team [13]. These allow each individual team member to participate with others, based on their understanding of their independence and interdependence from and to the team, allowing them to deal with the daily, weekly, monthly, etc., rhythms of the agile development cycles throughout the program/project [14].

The process of team and team member empowerment is a continual and active process, the form and efficacy of the empowerment process is determined by past, current, and on-going circumstances and events [15]. In essence, the empowerment process is an ebb and flow of independence and interdependence relationships that change throughout the agile development process, including each daily Scrum, each Sprint planning session, and each Lessons Learned session, throughout the entire agile development cycle of the program/project. Figure 3 illustrates this process.

In Figure 3, empowerment becomes an integral part of the overall agile development process, with evaluation of the team members' abilities, roles, independence and interdependence, based on the capabilities needed to be developed within a given Sprint, the honest evaluation of skills and abilities; i.e., how to develop the heartbeat, or development rhythm required for each development Sprint. Without an environment of Empowerment, the team has no real focus, since each team member does not have a sense of what they are individually responsible for, what the other team members are individually responsible for, and what communication is required throughout the Sprint development [16]. There will eventually be a breakdown of the team, a loss of efficiency, and the team will not be successful in their development efforts within cost and schedule constraints. Next we discuss the concepts of goal setting for an agile development project; project/program, team, and individual goals within the context of Locus of Control.

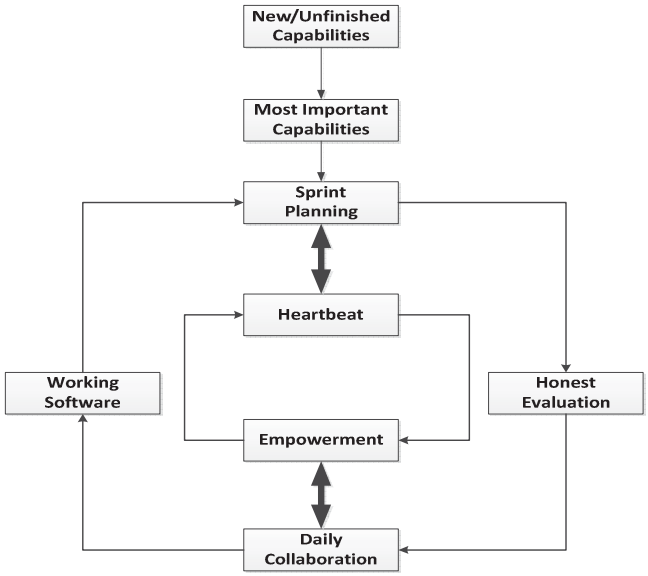


Figure 3 – Agile Development Process with Empowerment

4. Locus of Control in Agile Teams

As explained above, the very nature of agile software development is to create a loose structure both within each Sprint team and across the Sprint team structure. The purpose

of agile development is to allow developers, and subsequently the system being developed to adapt and change as requirements, features, capabilities, and/or development environment change over time (and they will change). However, this does not mean that there are not system-level, team-level, and individual-level goals at each point in time. In fact, it is more important in agile development to have well-defined goals as teams and individual developers write and test software, to ensure the software integrates and, more importantly, creates a set of capabilities and a system the customer wanted and is paying for. Customer and cross team collaboration and feedback at each level is crucial to allow the teams to adjust, either from customer needs or inter-team needs across the agile Sprint developments. Again, independence and interdependence is essential for overall successful development. Further refinement of the Empowerment concept is to define, for each individual developer, what things are within their own control, and those things are outside of their control, even if they affect the individual.

This notion of internal vs. external control is called "Locus of Control." Locus of control refers to the extent to which individuals believe that they can control events that affect them [17]. Individuals with a high internal locus of control believe that events result primarily from their own behavior and actions. Those with a high external locus of control believe that powerful others, fate, or chance primarily determine events (in this case other team members, other teams, the program/project manager, and/or the customer). Those with a high internal locus of control have better control of their behavior, tend to exhibit better interactive behaviors, and are more likely to attempt to influence other people than those with a high external locus of control; they are more likely to assume that their efforts will be successful [18]. They are more active in seeking information and knowledge concerning their situation.

Locus of control is an individual's belief system regarding the causes of his or her experiences and the factors to which that person attributes success of failure. It can be assessed with the Rotter Internal-External Locus of Control Scale (see Figure 4) [17]. Think about humans, and how each person, experiences an event. Each person will see reality differently and uniquely. There is also the notion of how one interprets not just their local reality, but also the world reality [19]. This world reality may be based on fact or impression.

For further thought let's then consider Constructivist Psychology. According to "The internet Encyclopedia of Personal Construct Psychology" the Constructivist philosophy is interested more in the people's construction of the world than they are in evaluating the extent to which such constructions are "true" in representing a presumable external reality. It makes sense to look at this in the form of legitimacies. What is true is factually legitimate and what is peoples' construction of the external reality is another form of legitimacy. In order to have an efficient, successful agile development team [20], each member must understand and accept their internal and external level of Locus of Control, as well as their Locus of Empowerment level. Figure 5 illustrates how this flows throughout the Sprint development cycles.

How an individual sees the external vs. internal empowerment drives their view of internal vs. external Locus of Control. During each development cycle, evaluations are made (whether they individual is aware of it or not) as to their internal and external Empowerment, and subsequent Locus of Control. Actions are determined, based on this self-assessment, and self-efficacy determination. Based on the results of their efforts, individuals, as well as the team, and the entire program/project re-evaluate the efficacy of the levels of internal vs. external Empowerment that are allowed, and adjustments are made. These adjustments to Empowerment levels drive changes in Locus of Control perception, which drives further actions. This process is repeated throughout the project/program. The manager must understand this process and make the necessary adjustment so that each individual can operate at their peak self-efficacy, as well as support team efficacy, providing the best atmosphere for successful development.

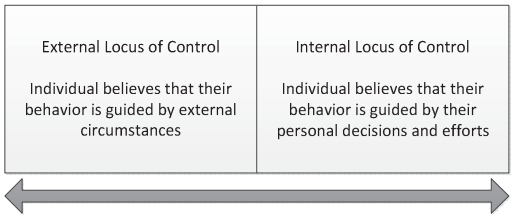


Figure 4 – Locus of Control Scale

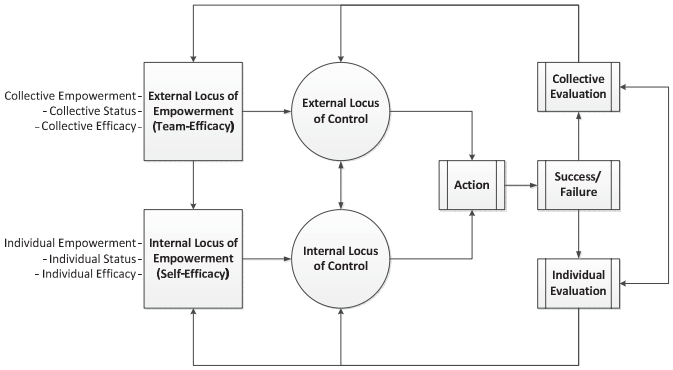


Figure 5 – Locus of Control within an Empowerment Cycle

5. Self Orgainization: The myths and the realities

One of the holy grails of agile development is self-organizing teams. Many software developers dream of having a team with complete autonomy, able to organize however works for them, completely without management involvement or interference. However, what most developers fail to realize is that given to their own devices, without training as to how to organize and what "organizing" actually means, most would fail miserably. Often, agile development efforts fail, even with efforts to educate the team about agile principles [21]. That is because the team doesn't fail because they don't

understand agile software development. It's because they don't understand human nature and the difficulties in taking a team of highly motivated, strong personalities, and get them to automatically give up their egos, pre-conceived notions, and past experiences, and embrace the agile team dynamics required to put together a highly successful agile development effort. We call this "Agile Team Dysfunctionality," and there are many common dysfunctions that plague improperly trained teams and team members. Figure 6 illustrates several of the most serious dysfunctions, most of which come both from basic human nature and from peoples experience working programs/projects in the past. Nothing drives failure of agile development like past failures. Teams that have experienced failure are hard pressed to through off their suspicions and embrace agile development processes, team dynamics, and the entire agile agenda fresh. Management must be cognizant of these dysfunctions and work within the teams to dispel them.

Inability to recognize or deal with agile team dysfunctions can de-stabilize the team(s) and de-rail the agile development process faster than anything else. Keeping a stable set of Sprints teams is important, as constantly changing out team members radically changes team dynamics, and effects both personal and team Empowerment and Locus of Control [22].

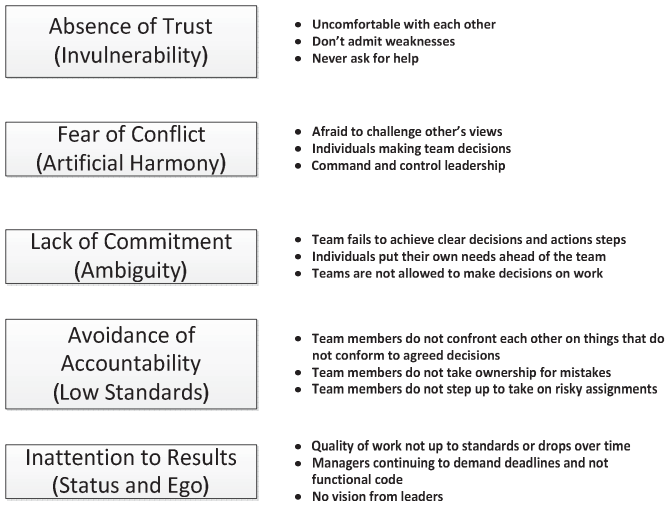


Figure 6 – Common Agile Team Dysfunctions

6. CREATING A STABLE TEAM MEMBERSHIP: CONTAINING ENTROPY

As previously discussed, it is vital to choose the right teams for any program/project, but it is even more important for agile development. Teams with stable memberships across Sprints is vital, as team members develop trust over time, gain an understanding of each members strengths, idiosyncrasies, and, with proper training, mentorship, and facilitation by the manager, settle into an agile development "rhythm" throughout the program/project. If the team has to integrate new members, efficiency will always suffer until the new team member is properly integrated into the rhythm. New expectations are created; the new person will most likely have an entirely different notion of Empowerment and Locus of Control than the previous team member, throwing the overall team out of balance. A stable team can be a trusting and efficient team [22]. There is generally an increase in commitment over time with a stable team [23]. In order to facilitate creation of stable agile sprint teams, the Agile Manager must recognize, understand, and know how to deal with the dysfunctionalities discussed in Section II. For each dysfunction, the Agile Manager must take on a role, or provide guidance that dispels the dysfunction and allows the team to move toward and independent cohesiveness between the team members [24]. Figure 7 illustrates the Agile Manger's role in dealing with classical agile team dysfunctions, creating a team that works together, in Empowered independence and dependence, to develop software in an efficient agile environment.

As depicted in Figure 7, for each of the agile team dysfunctions described in Figure 6, Figure 7 illustrates the Agile Manager's response required to eliminate the dysfunction and allow the agile development teams to function effectively and efficiently:

1. **Absence of Trust**: In order to build trust within the teams, the Agile Manager must always be willing to take the lead and prove to the team members that they will "roll up their sleeves" and do whatever is necessary to either get the program/project moving or to keep it moving along.
2. **Fear of Conflict**: Many developers are fearful of bringing up issues; not wanting to start controversy within the team. Many people, particularly strong introverts, may internalize the conflict, never bringing it up, but eventually the conflict will drive controversy between the developers, create a lack of trust, and may drive the team to withdraw from each other, destroying the collaborative nature of agile development teams. In order to diffuse these situations before they begin, the Agile Manager must be observant and cue in on body language and utilize the soft people skills like paying attention to changes in personal habits, language, friendliness, and other clues apparent between team members, and facial expressions to understand when such non-verbal controversies exist and work to resolve the conflict before they begin to negatively impact the development efforts.
3. **Lack of Commitment**: A lack of commitment to either the agile development team, or the agile process in general can destroy an agile program/project before it gets started. Observing a low quality of work, absenteeism, lack of willingness to communicate, or constantly seeming to be overwhelmed by the volume of work may be indications of a lack of commitment. The Agile Manager needs to understand the developer's reasons for the lack of commitment, clarifying for the developer what is expected, clearing up any misconceptions the developer may have. In the end, if the Agile Manager does not feel they have dispelled the lack of commitment, the

developer must be removed from the team or there is little hope for successful agile development. I know this sounds harsh, but agile only works if all parties have buy in to the agile development process.

4. **Avoidance of Accountability**: There may be issues getting developers to step up and take on rolls of responsibility within the agile teams because they are afraid that if they take responsibility for the team's activities during a given Sprint and there are problems, they will be punished. This lack of accountability needs to be dealt with in order for the Sprint development teams to develop a good business rhythm and operate effectively. It is up to the Agile Manager to confront issues, while not assigning blame or punishment, but working through difficult issues, helping each developer learn from the issues in order to solidify the teams and allow the developers to grow and mature as members of an agile development team. This will pay off in the future as each developer becomes more embedded in the agile process and learns to be effective in and excited about agile programs/projects.
5. **Inattention to Results**: Some developers like the agile team process because they feel they can just write code and let other people worry about the details, results, testing, etc. But, it is vitally important that the entire team focus on the results; working, error-free code with capabilities required for each Sprint that can be demonstrated. If any of the developers/team members are not focused on the results the team will never develop a good agile development rhythm. Also, one member being inattentive to details and results will breed mistrust between the members, reducing the effectiveness of the team(s). Therefore, the Agile Manger must keep the program/project vision in front of all developers and teams, making sure everyone is marching down the same path, ensuring that the collective outcomes of all the Sprint teams, across all of the Sprints integrates together and is heading toward a common, customer-focused goal.

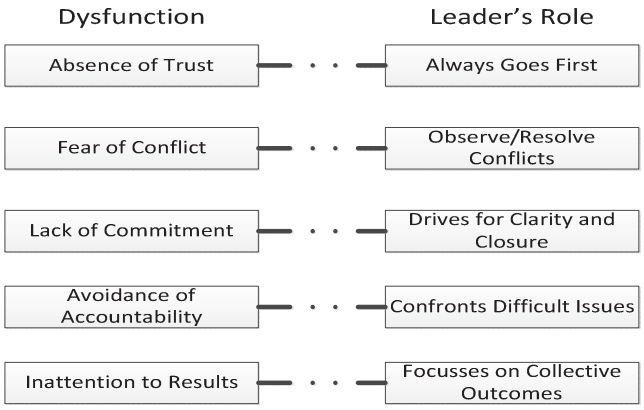


Figure 7 – The Agile Manager's Response to Team Dysfunctions

7. Conclusiong and Discussion: Creating an Environment of Learning and Growth

Agile development teams, at least the majority of teams, will be composed of developers at a variety of experience levels. Each member comes with their own strengths and weaknesses and should be provided an atmosphere that not only allows them to succeed, but to grow and learn, both from the experience of developing code for the program/project across the Sprints, but from each other as well. If facilitated correctly by the Agile Manager, the agile development program/project will allow opportunities for mentoring and learning. However, this must be designed into the Sprints, both in schedule and in capability distribution across the team members. Creating an atmosphere of mentoring, learning and creativity increases efficiencies, as the team progresses through the Sprints and help future programs/projects as well. Given the probable diversity of team members, the Agile Manager should make sure everyone has the opportunity and personal attitude of both mentoring and learning from each other. New software techniques brought by junior developers may be necessary for certain capabilities that older more experienced software developers may not be aware of. At the same time, junior developers should also bring an attitude of mentoring and learning, as the experienced developers can aide junior developers from going down disastrous roads already travelled by senior developers. In short, the atmosphere the Agile Manager must ***NOT*** bring to the agile development teams is illustrated in Figure 8.

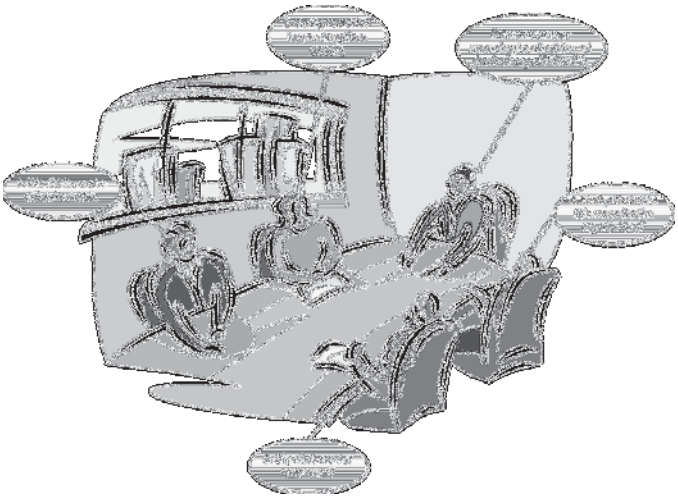


Figure 8 – The "Rigid" Agile Manager

8. References

1. Stanhope, D. S., Samuel B., I.,II, and Surface, E. A. 2013. Core self-evaluations and training effectiveness: Prediction through motivational intervening mechanisms. Journal of Applied Psychology, 98(5), 820-831.
2. Jewson, N. and Mason D. 1986. The Theory of Equal Opportunity Policies: Liberal and Radical Approaches. Sociological Review, 34(2).

3. Crowder, J. and Friess, S. 2014. The Agile Manager: Managing for Success. Springer Publishing, New York, NY, ISBN 978-3-319-09017-7.
4. Dirks, K. and Ferrin, D. 2002. Trust in leadership: Meta-analytic findings and implications for research and practice. Journal of Applied Psychology, 87, 611–628.
5. Crawford, A. 2008. Empowerment and organizational climate: An investigation mediating effects on the core self-evaluation, job satisfaction, and organizational commitment relationship. ProQuest Dissertations and Theses, 147.
6. Carless, S. 2004. Does psychological empowerment mediate the relationship between psychological climate and job satisfaction? Journal of Business and Psychology, 18(4), 405-425.
7. Eysenck, H. and Eysenck, S. 1969. Personality structure and measurement. San Diego, CA: Robert R. Knapp.
8. Crowder, J. and Friess S. 2013. Systems Engineering Agile Design Methodologies. Springer Publishing, New York, NY. ISBN-10: 1461466628.
9. Beck, K. 1999. Extreme Programming Explained - Embrace Change. Addison-Wesley, Boston, MA.
10. Thomas, K. and Velthouse, B. 1990. Cognitive Elements of Empowerment, Academy of Management Review, 15: 666-681. American Counseling Association. (2014). Code of ethics and standards of practice. Alexandria, VA.
11. Barnes, K. 2004. Applying self-efficacy theory to counselor training and supervision: A comparison of two approaches. Counselor Education and Supervision, 44(1), 56-69.
12. Spreitzer, G. M. 1995, Psychological Empowerment in the Workplace: Dimensions, Measurement and Validation, Academy of Management Journal, 38(5): 1442-1465.
13. Kernis, M. H. 2003. Toward a conceptualization of optimal self-esteem. Psychological Inquiry, 14, 1–26.
14. Lee, L. 1994. The empowerment approach to social work practice. New York: Columbia University Press.
15. Speer, P.W. 2000. Intrapersonal and interactional empowerment: Implication for theory. Journal of Community Psychology, 20(1), 51-61.
16. Skinner, E. A. 1996. A Guide to Constructs of Control, Journal of Personality and Social Psychology, 71(3): 549-70.
17. Rotter, J. 1966. Generalized expectancies for internal versus external control of reinforcement. Psychological Monographs, 80(1)
18. Bollen, K. 2001. Indictor: Methodology. In International Encyclopedia of the Social And Behavioral Sciences, ed. N. Smelser and P. Baltes, 7282-87. Elsevier Science, Oxford, UK.
19. Brooks, F. 1975. The Mythical Man-Month. Addison-Wessley, ISBN 0-201-00650-2.
20. Roope., K. 2003. Efficient Authoring of Software Documentation Using RaPiD7. icse, pp.255, 25th International Conference on Software Engineering (ICSE'03).
21. Lee, L. 1994. The empowerment approach to social work practice. New York: Columbia University Press.
22. Nichols, J. D. 2006. Empowerment and relationships: A classroom model to enhance student motivation. Learning Environments Research, 9(2), 149-161.
23. Larson, R., Walker, K., & Pearce, N. 2005. A comparison of youth-driven and adult-driven youth programs: Balancing inputs from youth and adults. Journal of Community Psychology, 33(1), 57-74.
24. Piotrowski, C. L. 2006. Quantum empowerment: A grounded theory for the realization of human potential. (Order No. 3240834, Cardinal Stritch University). ProQuest Dissertations and Theses, pp. 358.

An Architecture for Dynamic Self- Adaptation in Workflows

Sheila Katherine Venero Ferro[1] **and Cecilia Mary Fischer Rubira**[1]
[1]Institute of Computing, University of Campinas (UNICAMP), Campinas, Sao Paulo, Brazil
ra144653@students.ic.unicamp.br, cmrubira@ic.unicamp.br

Abstract - *Today several organizations use many kinds of Process-Aware Information Systems to support their processes. A typical example of such systems is Workflow Management Systems. However, due to the complexity of business processes and its continuously changing environment, there is an inevitable need to expand the dynamic behavior of these computational solutions. One of the drawbacks of the workflow platforms is that they usually cannot dynamically adapt their processes. The aim of this research is to develop an architecture for workflow management systems that provides means of flexibility to dynamically adapt the workflows during runtime. In order to validate our solution, a case study was conducted in nursing processes based in a real diagnosis scenario to show a practical applicability of the proposed adaptive architecture. Preliminary results have shown that the architecture successfully supports business logical adaptations.*

Keywords: Workflow Management Systems; dynamic process adaptation; Process-Aware information systems; software architecture.

1 Introduction

Due to the continuous dynamism of the business environment, organizations should be prepared to respond different situations and unpredictable events in order to maintain leadership. To do so, many kinds of Process-Aware Information Systems are used by the organizations to support their processes. Workflow Management Systems (WFMSs) are typical examples of such systems [1] which partially or totally automate business processes. WFMSs define, create, and manage the execution of workflows through software, executing one or more workflow engines that allow process definition, user interaction, and provide means for invoke IT tools and other applications [2].

Traditional Workflows Management Systems usually work with well-structured processes and typically for predictable and repetitive activities [3]. However, modern processes often are required to be flexible in order to reflect foreseeable or even unforeseeable changes in the environment. Thus, WFMSs face some limitations in means of flexibility; they cannot support dynamically changing the business process or just support them in a rigid manner [1]. Dynamic workflow changes can be either in a single instance or an evolutionary change in the process schema, so the adaptations can be at the instance level or at the type level [4].

Workflow technology can deliver the right information to the right person and at the right time reflecting all the changes. And if this technology could adapt the processes dynamically according to the environment or context they also would help in the decision-making process and certainly improve the efficiency of business processes to respond to unexpected changes. As an illustrative example, considering a nursing process, a system can alert the nurse that a patient is allergic to a particular drug, suggest other similar drugs or redefine on-the-fly the care plan of a patient, because when planning (at design time) it is almost impossible to prevent all the situations since the number of different possibilities is too high.

Lately adaptability in workflow technology is one of the hot topics in the academic world [5]. Nevertheless, just a few of approaches treat adaptation at the business logic level. Most of the approaches deal adaptation at the technological or performance level, treating exceptional behavior caused by errors in the application or errors in the infrastructure or middleware components on which the process runs. This paper is focused on dynamic adaptation in workflows at the business logic level, treating exceptional behaviour caused by the result of a breach of a business rule, a constraint violation, a data issue, or an unexpected business behavior.

Most approaches support changes in the environment, failures, variations and exception using policy/rule-based frameworks [3, 6, 7, 8, 9, 10] and explicitly represent paths or schemas. Many of them use ECA (Event-Condition-Action) rules to define constraints. These kinds of approaches work well with well-defined business process; however they are not suitable for more complex and dynamic processes or for weakly structured or non-routine process. Other modern approaches uses knowledge-based techniques [11,12, 13, 14, 15, 16, 17], case-based reasoning [18] or ontology-based reasoning [19, 20, 21]. These kinds of approaches help with unpredictability and uncertainty of business processes, and with non-routine processes and provide human thinking and decision-making techniques that provide sufficient flexibility and adaptability for business process [13].

We believe that the usage of autonomic computing principles along with some cognitive capabilities can satisfactory cope with foreseen and unforeseen changes in

business process and permit adaptation at runtime. In this paper, we propose an architecture based on the MAPE-K reference model that provides flexibility for workflow management systems to dynamically adapt business processes to suit new conditions at runtime. This paper is organized as follows: Section 2 describes the background of this work. Section 3 shows the proposed solution. Section 4 presents the case study and finally Section 5 presents the conclusions.

2 Background

This section describes the theoretical foundation used in the proposed solution.

2.1 Workflow Management System (WFMS)

A workflow Management system can be defined as a software system that defines, creates, and manages the execution of workflows. It possesses one or more workflow engines capable to interpret the process definition, enable user interactions, and invoke different IT Tools and applications [2]. The workflow execution has a specific order of execution according to business logic [22].

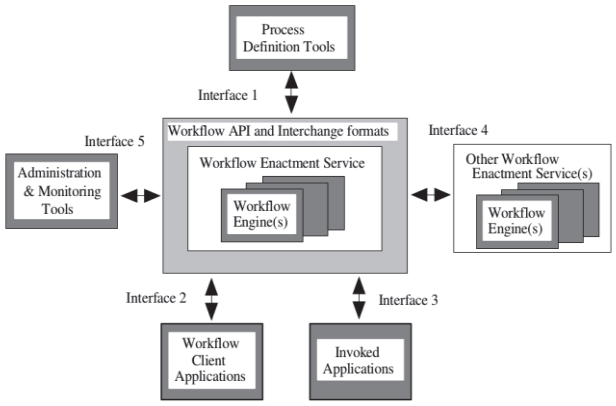


Figure 1. Workflow Reference Model - Components & Interfaces [22]

Figure 1 shows The major components and interfaces within the architecture according to [22], which are described below:

The workflow enactment service provides a runtime environment in which one or more workflow management engines create, manage, and execute workflow instances and interact with external resources by means of workflow application programming interface. Several functions can be handled by the workflow engine, including interpretation of the process definition, control of process instances, navigation between process activities, sign-on and sign-off of specific users, identification of work items for user attention and an interface to support user interactions, maintenance of workflow control data and workflow relevant data, passing workflow relevant data to/from applications or users, an interface to invoke external applications and link any workflow relevant data, supervisory actions for control, administration, and audit purposes. It interacts with the external resources through the interfaces [22].

Process Definition Tools are different tools that define and model business process and its activities, translating them from the real world to a formal computerized representation. These tools may be supplied as part of a workflow product or as a separate. The interface between those tools and the runtime workflow enactment service is called the process definition import/export interface [22].

Workflow Client Applications are activities that require involving human resources. The interface between these client applications and workflow engine interacts with the Worklist handler, responsible for organizing the user interaction with the process instance. It is the responsibility of the Worklist handler to choose and advance each element of the Worklist [22].

Invoked Applications are other potential applications without user interaction in a heterogeneous product environment. The interface allows the workflow engine to activate a tool to perform a particular activity, for this reason, there must exist a common format to transfer data among them [22].

Administration and monitoring tools provide operations such as user management, role management, audit management, resource control, process supervisory functions, etc. [22].

There should be interoperability functions that provide communication between heterogeneous workflow systems [22].

2.2 Self-Adaptive Systems

A self-adaptive system adjusts its artifacts or attributes in response to changes. To accomplish its goal, it should monitor the software system (self) and its environment (context) to detect changes, make decisions, and act appropriately. The basis of self-adaptive software is the adaptation of dynamic/runtime changes [23]. This kind of software tries to fulfill its requirements at runtime in response to changes [24]. These systems make decisions on their own, using high-level rules and policies. They constantly check and optimize their status and automatically adapt themselves to changing conditions, keeping the system's complexity invisible to the user and operators.

According to [24], to contemplate their goals they should have some features known as self-* properties: *Self-configuration*, it is the ability of automatic configuration of components according to high-level goals. *Self-optimization*, it is the ability of automatic monitoring and control of resources to ensure the optimal functioning. The system may decide to initiate a change to the system proactively in an attempt to improve performance or quality of service. *Self-healing*, it is the ability of automatic detection, diagnosis, correction, and recovery of faults. *Self-protection*, it is the

ability of identification and protection of malicious attacks but also from end-users who inadvertently make software changes.

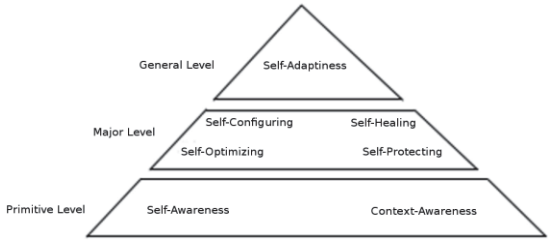


Figure 2. Hierarchy of the Self-* Properties [23]

[23] discusses these properties along with some others and provides a unified hierarchical set, Figure 2 describes a hierarchy of self-* properties in three levels. In this hierarchy, self-adaptiveness is in the general level which is decomposed into major and primitive properties at two different levels. The primitive properties are *Self-awareness* and Context-Awareness. Self-awareness means aware of its self-states and behaviors according to [25] and *context-awareness* means aware of its context its operational environment according to [26].

2.3 MAPE-K

MAPE-K is the reference model for autonomic control loops suggested by IBM [27]. It is a logical architecture that defines the main architectural blocks for building an autonomic manager either in a monolithic or distributed approach.

The MAPE-K derives its name from the main tasks in the feedback control loop of self-adaptive systems [27]:

- *Monitor*, it collects information from the managed resources. The monitor function aggregates, correlates, and filters the information until it determines a symptom that needs to be analyzed.
- *Analyze*, it performs complex data analysis and reasoning on the symptoms provided by the monitor function. This analysis is made with stored knowledge data. If there is a need of changes, the request is logically passed to the plan function.
- *Plan*, it structures the actions needed to achieve goals and objectives. The plan function creates or selects a procedure to enact a desired alteration in the managed resource to cope with the new situation.
- *Execute*, it carries out the adaptation, changes the behavior of the managed resource using effectors based on the actions recommended by the plan function.
- *Knowledge*, standard data shared among the monitor, analyze, plan, and execute functions. The shared knowledge includes data such as topology information, historical logs, metrics, symptoms, and policies. Knowledge is created by the monitor part while execute part might update the knowledge.

2.4 Intelligent Agents

An agent is an autonomous software entity situated in some environment where it takes autonomous actions to achieve their goals. They are capable of making decisions to proactively or reactively respond changes in its environment in real-time [28].

According to [29], agents are *autonomous* (operates without direct human intervention and control their internal states), *social* (interact with human and other agents), *reactive* (perceive changes in the environment and responds to it in a timely fashion), *proactive* (takes the initiative to satisfy its goals, goal-directed behavior).

2.5 Flexibility in Process

[30] defines flexibility as the ability to yield to change without disappearing, without losing identity. In processes, flexibility is defined as the ability to deal with both foreseen and unforeseen changes, adapting or varying the affected parts of the business process and maintaining the essential form of the parts not impacted [31].

According to [31] the flexibility types can be classified as: *Flexibility by Design*, design alternative execution paths within a process model at design time. The selection of the most appropriate path is made at runtime for each process instance. *Flexibility by Deviation*, a process instance might temporarily deviate at runtime from the execution path prescribed in order to deal changes in their environment without altering the process model. *Flexibility by Underspecification* is the ability to execute an incomplete process model at runtime. Placeholders are variable points marked as underspecified, where is not possible design the activities because the lack of information. E. g. Late binding, late modeling, etc. *Flexibility by Change* is the ability to modify a process model at runtime. This means to migrate all current process instances to the new process model.

3 The Proposed Solution

To enable dynamic adaptation in workflows, the proposed architecture (Figure 3) uses the reference model for autonomic control loops MAPE-K which provides to the workflow autonomic properties such as self-configuration, self-healing, self-optimizing, and self-protecting. With these self* properties along with self-awareness and context-awareness, the workflow fulfills the requirements to be self-adaptive and dynamically respond to changes at runtime.

An important quality attribute of the architecture is the separation of concerns between the business logic and the application logic. The adaptation layer is separated from the workflow engine, therefore the workflow adaptation is performed in a meta-level, and consequently it is transparent

to the workflow engine, so we do not modify the structure of any workflow engine. In addition, it gives us an easy and independent way to manage changes in the business process. This paper is focused in the adaptation layer.

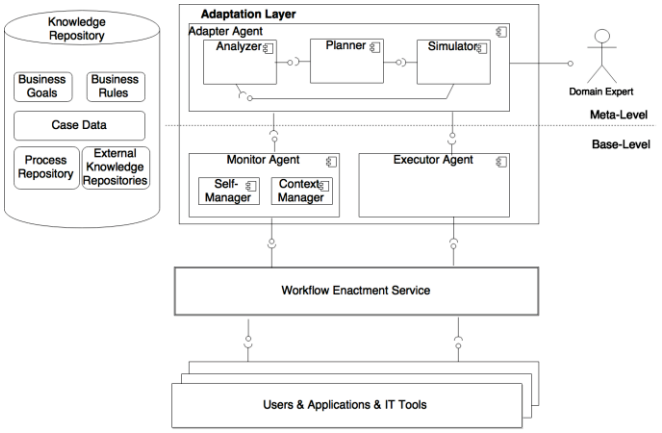


Figure 3. Proposed Architecture

The adaptation layer uses intelligent agents which make the workflow autonomous, proactive, reactive, and goal oriented. Since these operate without direct human intervention, can communicate to each other's, perceive the environment, respond to changes, and take initiatives to achieve the goals. To achieve their goals, agents need to use knowledge to reason about its current state and the environment, to generate plans and execute them, for this reason, a shared centralized knowledge repository is provided.

The knowledge repository is the core element of our architecture, which centralizes all the information related to the business goals, business rules or policies, case data, a process repository (set of activities that can be chosen at runtime), and other external knowledge repositories related to the business domain. According to this information, the agents monitor, analyze, make decisions, and plan new activities. The information should be represented in the form of declarative knowledge to do logical inferences and solve problems.

To distribute the roles, the adaptation layer has a monitor agent, adapter agent, and executor agent, consecutively the adapter agent comprises the analyzer, the planner, and the simulator separated according to their capabilities or functions. All the agents share the knowledge repository.

The monitor agent has a self-manager and a context manager. The self-manager acquires the current status of the executing instance. The context manager interprets the acquired context information during the execution of an instance and represents it in a context model interpretable for the monitor. The context manager can also use some process mining techniques for discover other context information from the execution environment in order to update the context model.

The monitor agent continuously evaluates the current state of the process instance and its context provided by the context manager during all the process execution until it determines a symptom that need to be analyzed, that could be caused by the result of a breach of a business rule, a constraint violation, an unexpected data value or output or any other unexpected business behavior. This symptom is delivered to the analyzer.

The analyzer observes, identifies, and reasons over the situation according to the business rules and goals and diagnoses the situation. The analyzer counts with artificial intelligence techniques to make the data analysis, reason and make real-time decisions, the analysis is influenced by all the knowledge base provided. It is responsible for identify potential adaptations during the process execution. If adaptations are required, the planner is activated.

The planner creates or selects activities from the process repository, it checks if the detected situation has a solution, if not it makes some inferences of a possible solution also it counts with artificial intelligence planning techniques to reconfigure the activities, according with preconditions and postconditions, interdependencies between activities, business rules, etc. It reformulates and reconfigures the process at a high level of abstraction. The solution can be a single activity or a complex process. It also employs predictions techniques to make predictions about the planned activities, to watch their impact.

The simulator reproduces the activities and verifies if it has to do some extra changes in the process or if there are some inconsistencies if so, this new situation is passed to the analyzer. Otherwise, the solution is suggested to a domain expert, who will approve the proposal or further adapt it, after the solution will be learned. Depending on the particular application domain, processes can be only altered by the supervision of an expert.

The executor agent makes the process definition: creates a computerized representation of the process that will be interpreted by the workflow engine.

In order to validate our proposal, the next section presents a case study based in a real nursing case.

4 Case study: Nursing Process

This section presents an evaluation of the proposed architecture to prove its effectiveness and feasibility adapting workflows at runtime. Thus, the main goal of this case study is to analyze if the proposed architecture contemplates dynamic adaptation in workflows at runtime in the context of a nursing domain. We chose the nursing domain because the nursing process is typical example of a flexible process. In the nursing practice, nurses monitor patients during their treatments, execute different tasks according to each patient situation and react to unexpected situations. For these reasons explained above, we believe that a nursing process is a very

good example of a workflow with unexpected situations and changes, consequently is excellent to evaluate the proposed architecture.

4.1 Case Study Context

Nursing is a service provided to humans focused on assistance and patient care in different situations related to his health [32]. Nurses adopt practices based on scientific knowledge and develop a systematization of their processes. Thanks to this systematization, the nursing profession achieved its professional autonomy. With this autonomy, nurses can give a diagnosis, different from a medical diagnosis. Thus, nurses should have necessary knowledge and experience in order to take care a patient and make decisions about what treatment or procedures they may do.

A medical diagnosis deals with disease or medical condition. A nursing diagnosis focuses on the person and their physiological and/or psychological response to actual or potential health problems and life processes [33].

According to the Federal Nursing Council Resolution No. 358/2009, the nursing process is a methodological tool that guides professional care and nursing documentation of professional practice. The steps for realization of the *nursing process* (Figure 4) are: collect nursing data, nursing diagnosis, nursing planning, implementation, and evaluation of nursing. These five steps are interrelated, interdependent, and recurring [34].

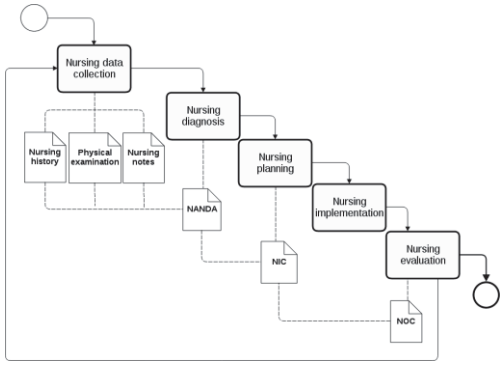


Figure 4. Nursing Process

The first step, *collect nursing data*, is collect relevant data about the patient, family, and community, to identify needs, problems, worries, and patient's human reactions [35]. This is basically through anamnesis and physical examination. The *nursing diagnosis* is the process of interpretation of the data collected, the decision making process based on the concepts of nursing diagnoses which form the basis for selecting interventions and expected results [35]. The *nursing planning* consists in determining priorities between diagnosed problems, setting the expected results for each problem and their respective prescriptions in an organized way. According of the expected results, a set of nursing interventions is planned. The i*mplementation* of the nursing plan attends the whole process to minimize risks, solve or control a problem (nursing diagnosis), assist in daily activities and promote health. Nurses should be constantly aware of both patient responses as well as their own performance because the human being is unpredictable and requires constant monitoring [36]. The *evaluation* is a deliberate, systematic, and continuous process of verification. It consists in follow the patient responses to the prescribed care through observation notes in the respective medical record. The nurse evaluates the progress of the patient, establishes corrective measures, and if necessary, revises the care plan.

The nursing process helps nurses to make decisions, predict, and assess consequences. It improves the nursing capacity to solve problems, make decisions, maximize opportunities and resources to form habits, and increase their expertise.

4.2 Research Questions

The study answers the following questions.

RQ1: Does the architecture provide means to adapt the workflow under new circumstances or unexpected behaviors during the workflow execution?

RQ2: What kinds of changes does the architecture support?

4.3 Case selection and units of analysis

In case studies, the case and the units of analysis are selected intentionally [37]. So, the main selection criterion was that the nurse case scenario should present unexpected situations or events and provide us a good description of the events in the nursing process. But also the selected unit was limited to its availability. The unit for analysis was selected because its extreme behavior, the real-life medical scenario analyzed is described in [38].

4.4 Data Analysis

The data was collected through a documentary analysis. The medical scenario was reconstructed in order to get the sequence of events to be mapped into the nursing process. We made a time -series analysis in order to denote the set of events that happen over the time and executed a simulation of a process instance (a nursing scenario) in order to verify if the architecture supports all unexpected situations that happen in this nursing scenario. The domain was modeled as follows:

Knowledge Repository

It was populated with relevant information of the nursing domain:

- Normal rates for vital signs → Business Rules
- NANDA (North American Nursing Diagnosis Association) → Business Rules

- NOC (Nursing Outcomes Classification) → Business Goals
- NIC (Nursing Intervention Classification) → Process Repository
- Medical Record → Case Data
- Drugs information's, evidence- based medicine → External Knowledge Repositories

Context information

- New symptoms (Adverse reactions, allergies,etc.)
- Abnormal vital signs
- Contraindications
- Active health problems
- Active medications

The procedures of each part of the adaptation layer were:

Monitor agent: Continuously analyzes the case data until some unexpected events happen; new context elements appear or are identified by the context manager.

Adapter Agent: Divides tasks between the analyzer, planner, and simulator.

- **Analyzer:** The analyzer observes, reasons about the events, characterizes the situation according to the NANDA taxonomy, search for the defining characteristics, analyze the medical record, and finally give the nursing diagnoses for this situation.
- **Planner:** With this diagnostic, the planner searches interventions (NIC) for the diagnoses in order to create a care plan and structures the activities needed to achieve the goals and objectives (NOC).
- **Simulator:** Finally, the simulator simulates all the process and search for potential inconsistencies or dangerous situations. If some problem is detected, the process is replanned. Finally it will be suggested to the nurse. After her approval, the process is stored in the process repository.

Executor Agent: After that process, the executor agent will make the process definition to be delivered to the workflow engine.

This cycle is repeated several times in the execution of the nursing scenario and this information is all the time stored in the case data.

4.5 Threats of validity

- *Construct Validity*, The threat of validity is that the chosen nursing scenario is a real life case documented in [38], so it counts with expertise of nurses.
- *External Validity*, it is not possible to say that the study case is exhaustive. Other studies for different processes and domains have to be performed.
- *Internal Validity*, the simulated scenario is a real-life nursing case chosen because it shows an extreme nursing case with many unexpected circumstances. The proposed workflows were validated by a nurse.
- *Reliability*, this study presents a limitation related to its results, which will be considered only as evidence.

4.6 Results

RQ1: Does the architecture provide means to adapt the workflow under new circumstances or unexpected behaviors during the workflow execution?

Based on the simulated situations during the execution of the nursing scenario, it was shown that the architecture has means to adapt the processes at runtime: the monitor agent, the adapter agent make possible the process adaptation using the knowledge repository and compose the process in high level of abstraction to finally the executor agent make the process definition at runtime in order to be interpretable for the workflow enactment service. The workflow adaptation is only possible with a vast knowledge repository. The dynamic adaptation is imperceptible to the workflow enactment service because it is being made in a higher level of abstraction.

RQ2: What kinds of changes does the architecture support?

The architecture supports both foreseen and unforeseen changes in process instances. The foreseen changes are previously modeled in the process repository at design time, so the architecture provides flexibility by design. The unforeseen changes are supported by the adaptation layer at runtime. The nursing planning is a step in which there is insufficient information at design time, so as the architecture provide means to execute an incomplete process model, the architecture provides flexibility by underspecification. Flexibility by deviation is also supported by the architecture because during the process execution we deviate several times from the initial plan to cope changes in the context over the time.

Thus, the proposed architecture contemplates dynamic adaptation in workflows at runtime in the context of a nursing domain. We must mention that for this domain, it is very important to know that every taken decision (diagnosis, interventions, and activities) by the agents, it is first suggested to the nurse (domain expert) who approves it or change it. After its acceptance, the suggested workflow is saved as an experience then the workflow is created to be interpretable for the workflow engine. The suggested information helps medical professionals to identify unexpected situations and make better clinical decisions, nursing diagnosis, and nursing interventions.

5 Conclusions

In this paper was proposed an architecture for workflow systems, we focused in the adaptation layer that permits dynamic adaptation in workflows. The adaptation layer is based in the autonomic control loop MAPE-K that provides the means for adaptation, each module of the MAPE-K is represented by an intelligent agent that follows business rules in order achieve the business goals. The agents are capable to make decisions in order to deal with unexpected changes in both "the self" and "its context", proactively work together to achieve the business goals and learn from its experience. We believe that our approach is simple but potent.

In order to validate the solution, a case study was made in the nursing domain. An extreme nursing case scenario was simulated in the architecture in order to illustrate the use of our approach. The proposed architecture has showed its capacity to support different situations and dynamically adapt the process according to unexpected circumstances. Therefore, the architecture supports dynamic adaptation at runtime.

Our approach is part of an ongoing work. As such, much remains to be done. As future work, we plan to develop a system that performs dynamic adaptation in workflows in order to empirically evaluate the proposal.

Acknowledgment

Thanks to the National Council for Scientific and Technological Development – CNPq – for financial support.

References

[1] W. M. P. Van Der Aalst, "Process- Aware Information Systems: Design, Enactment, and Analysis," Wiley Encycl. Comput. Sci. Eng. , pp. 1–31, 2009.
[2] M. Weske, Business process management: concepts, languages, architectures. 2007, no. Second Edition. 2007.
[3] S. Deng, Z. Yu, Z. Wu, and L. Huang, "Enhancement of workflow flexibility by composing activities at runtime," … 2004 ACM Symp. …, pp. 667–673, 2004.
[4] M. Reichert, S. Rinderle, and P. Dadam, "On the Common Support of Workflow Type and Instance Changes under Correctness Constraints," 2003.
[5] W. M. P. Van Der Aalst and S. Jablonski, "Dealing with workflow change: Identification of issues and solutions," Comput. Syst. Sci. Eng. , vol. 15, no. 5, pp. 267–276, 2000.
[6] R. Müller, U. Greiner, and E. Rahm, "AgentWork: a workflow system supporting rule-based workflow adaptation," Data Knowl. Eng. , vol. 51, no. 2, pp. 223–256, Nov. 2004.
[7] R. Romeikat, B. Bauer, T. Bandh, G. Carle, H. Sanneck, and L. C. Schmelz, "Policy-driven Workflows for Mobile Network Management Automation," pp. 1111–1115, 2010.
[8] A. Agrawal, "Semantics of business process vocabulary and process rules," Proc. 4th India Softw. Eng. , pp. 61–68, 2011.
[9] G. Russello, C. Dong, and N. Dulay, "Personalising Situated Workflow Systems for Pervasive Healthcare Applications," 2008.
[10] J. M. Bernal and P. Falcarin, "Dynamic context-aware business process: a rule-based approach supported by pattern identification," Proc. 2010 …, pp. 470–474, 2010.
[11] M. Wang, H. Wang, and D. Xu, "The design of intelligent workflow monitoring with agent technology," Knowledge-Based Syst. , vol. 18, no. 6, pp. 257–266, Oct. 2005.
[12] Y. Dai and J. Wang, "Variation knowledge-based approach to handling business process changes," 2006, pp. 693–700.
[13] M. Wang and H. Wang, "From process logic to business logic—A cognitive approach to business process management," Inf. Manag. , vol. 43, no. 2, pp. 179–193, Mar. 2006.
[14] Y. Qu, X. Sheng, and W. Jiao, "A Multi-Agent Based Model of Workflow Management," 2006, pp. 8–12.
[15] K. Lee, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes, "Workflow Adaptation As an Autonomic Computing Problem," in Proceedings of the 2Nd Workshop on Workflows in Support of Large-scale Science, 2007, pp. 29–34.
[16] P. Chakravarty, "An Event-Driven Approach for Agent-Based Business Process Enactment," vol. 5, pp. 1269–1271, 2007.
[17] W. Duo, L. Yi, L. Wenhui, J. Qi, and Y. Rongqing, "Intelligent Multi-Agent Based Information System of Business Process Management," 2008 IEEE Pacific-Asia Work. Comput. Intell. Ind. Appl. , pp. 469–473, Dec. 2008.
[18] Y. Stavenko, N. Kazantsev, and A. Gromoff, "Business Process Model Reasoning: From Workflow to Case Management," Procedia Technol. , vol. 9, pp. 806–811, Jan. 2013.
[19] J. Dang, A. Hedayati, K. Hampel, and C. Toklu, "An ontological knowledge framework for adaptive medical workflow. ," J. Biomed. Inform. , vol. 41, no. 5, pp. 829–36, Oct. 2008.
[20] A. Z. Abbasi and Z. a. Shaikh, "A Conceptual Framework for Smart Workflow Management," 2009 Int. Conf. Inf. Manag. Eng. , pp. 574–578, 2009.
[21] S. Mitsch, W. Gottesheim, F. H. Pommer, B. Pröll, W. Retschitzegger, W. Schwinger, R. Hutter, G. Rossi, and N. Baumgartner, "Making Workflows Situation Aware - An Ontology-driven Framework for Dynamic Spatial Systems," pp. 5–7, 2011.
[22] D. Hollingsworth, "Workflow Management Coalition: The Workflow Reference Model," 1995.
[23] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Trans. Auton. Adapt. Syst. , vol. 4, no. 2, pp. 1–42, 2009.
[24] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," vol. 36, no. 1, January, pp. 41–50, 2003.
[25] M. G. Hinchey and R. Sterritt, "Self-managing software," Computer (Long. Beach. Calif). , vol. 39, no. 2, pp. 107–109, 2006.
[26] M. Parashar and S. Hariri, "Autonomic Computing: An Overview," Hot Top. Lect. Notes Comput. Sci. 3566, pp. 257–269, 2005.
[27] IBM Corporation, "An Architectural Blueprint for Autonomic Computing, Technical Whitepaper (Fourth Edition)," no. June. 2006.
[28] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd Edition, London, UK, John Wiley & Sons, 2009.
[29] M. Wooldridge, Intelligent Agents Theories, Architectures, and Languages, Springer-Verkag, 1995.
[30] G. Regev and A. Wegmann, "A Regulation-Based View on Business Process and Supporting System Flexibility," Proc. CAiSE 05 Work. Bus. Process Model. Dev. Support BPMDS 05, no. Section 2, pp. 91–98, 2005.
[31] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Van der Alst, "Process flexibility: A survey of contemporary approaches," Lect. Notes Bus. Inf. Process. , vol. 10, no. Part I, pp. 16–30, 2008.
[32] W. A. Horta, Processo de Enfermagem. São Paulo: EPU, 1979.
[33] NANDA International, Nursing diagnoses: definitions and classifications, Wiley Blackwell, 2010.
[34] Brasil, Resolução COFEN nº 358 de 15 de outubro de 2009. Conselho Federal de Enfermagem. Rio de Janeiro; 2009.
[35] M. C. Tannure, A. M. Gonçalvez, SAE: Sistematização da Assistência de Enfermagem. Guia Prático. Rio de Janeiro, Brasil, 2010.
[36] R. Alfaro-Lefreve, Aplicação do processo de enfermagem: promoção do cuidado colaborativo. 5 ed. Porto Alegre: Artmed, 2005.
[37] P. Runeson, M, Host, A. Rainer, B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, 1st Edition, John Wiley & Sons, New Jersey, USA, 2012.
[38] J. R. Mancia, "Revista Brasileira de Enfermagem," Rev. Bras. Enferm. , vol. 53, no. 1, pp. 5–6, 2007.

Distributed Agile Development: A Survey of Challenges and Solutions

Harneet Kaur, Hisham M. Haddad, and Jing (Selena) He
Department of Computer Science, Kennesaw State University, Kennesaw, GA, USA
fharneet@students.kennesaw.edu, {hhaddad, jhe4}@Kennesaw.edu

Abstract-*Global market, constant pressure, and global talent are major powers that lead to distributed teams. But this is not an easy task; companies have to deal with several non-trivial geo-political constraints, such as the willingness of their employees to relocate, the costs of such relocations, procedural constraints such as work permit issues, and others. To circumvent this, companies set up their offices in multiple locations and hire local employees who work with their colleagues at different geographical locations. Agility at Scale 2012 survey found that 57% of the respondents were having geographically dispersed teams. Having your resources in different locations may make implementing certain agile processes harder. However, there are ways of working around it. This paper is investigates the challenges faced by distributed agile teams and proposes solutions to address these issues through an exploratory literature review. The proposed solutions will help to build successful agile teams.*

Keywords: Distributed agile development, Agile practices, Outsourcing, Distributed agile challenges, Distributed agile solutions.

1. Introduction

In the most recent decade, there is a continuous investment to transform local markets to global market. Many different challenges have to be managed like additional faults in software projects and lack of sufficient assets. Software organizations use Distributed Software Development (DSD) amenities to deal with these problems. These amenities help minimize expenses and the way into skilled labour. Their primary goal is to build up worth goods at reduced prices than the co-located developments by enhancing resources. At times, the quest for extreme benefit takes organizations to seek outside solutions in different nations and this is referred to as Global Software Development (GSD).

Software is created at multiple locations, in multiple cultures and in worldwide dispersed locations. The managers, executives, and engineers have to face many challenges at different stages of the development process. These challenges may be social, cultural, or technical. This influences the manner in which software is planned, executed, and conveyed to the customer.

Agile methods work well in notably vibrant industry and IT environment. Organizations are restless in looking for talent and skills accessible at easier rates. And hence the desire to outsource the development process to these countries. The aim of this paper is to understand the challenges faced by the geographically dispersed agile teams and propose practices that can be used to overcome these challenges.

The paper is organized as follows: Section 2 highlights distributed agile development. Section 3 discusses the challenges faced by geographically dispersed agile teams. Section 4 presents solutions for these challenges. The section 5 concludes the paper.

2. Distributed Agile Development

Agile development came into existence in 2001 [1] and was considered to be the foundation to change the software development practices. This was accomplished by mediating the risk of altering needs and evolving technologies. Agility intends to strip away the complexity associated with traditional development. Thus concentrates on its deciding objective to accelerate the project due dates, lift up the brisk reaction to evolving situations and changes in client necessities etc.

Distributed development is considered to be an unavoidable truth for numerous agile teams. Majority of agile methods presume that the teams to be placed in single room but unfortunately this does not go with the real world situations in which the agile teams are distributed globally throughout the world. The factors that give rise to dispersed teams are the following:

1. *Global Market:* Business market is expanding at a very fast rate and when some new business steps in, it has to match its standards. For that purpose it needs to advance the knowledge in those markets with the help of amalgamation and setting up or gaining subsidiaries situated in those markets.

2. *Global Talent:* In this competitive world, companies are hiring skilled and experienced employees. There are some factors that lead to distribution of teams such as: Willingness of employees to relocate, Availability of Work Visa, and Cost of relocation.

3. *Reducing Costs:* Nowadays companies are outsourcing to areas with economical development rates to reduce the development cost. It is estimated that 25% cost savings are there if the service providers are located offshore than domestic.

There are several Agile stories reported in the literature [2, 3, 4, 5]. The "Agility at Scale 2012" survey [6], illustrates how distributed were the teams on the successful projects: 49% of the agile teams had team members spread out through the same building; 29% of the team members were working within the driving distance; 30% of the agile team members were working from home; and 52% indicated that some of their agile teams had team members that were far located.

These facts illustrate that software companies do not go with the idea of whole agile team working together in one room. So, there is a strong requirement to combine Agile practices with distributed development practices. Combining these approaches introduces some challenges which are discussed in the following section.

Organizations that choose to be both distributed and Agile have different ways of implementing that through the team structure: *Isolated*, *Semi-integrated*, and *Integrated* [7]. With *Isolated* structure, the organization may decide to separate all the functionality at a particular location to form isolated teams. The benefit of this approach is that numerous issues related to distributed teams are averted. With *Semi-integrated* structure, every location develops a team and only the set of overlapping features with other teams are dealt. This structure promotes further sharing of knowledge and lesser knowledge silos. Only drawback of this structure is dealing with communication problems. With *Integrated* structure, the team is composed of members from diverse locations. Although this maximizes knowledge sharing, it increases the possibility of miscommunication due to lack of team cohesiveness.

3. The Challenges

Although distributed teams are considered to be more effective than co-located teams because of the reduced cost, global talent, and others, there are certain loopholes in this approach. The following are the challenges faced by distributed agile teams:

3.1 Documentation

Agile teams do not give importance to the documentation. This may affect the distributed teams as they will miss some details about the project and hence their understanding about the project will suffer.

3.2 Pair Programming

Agile development uses pair programming in which two members of the team work on the same code side by side. This approach is totally impossible in distributed environment. Hence distributed groups will have to find some other similar methodology.

3.3 Different Work Hours

Sometimes, there come situations when team members are located in different time zones and their working hours doesn't match. Hence their working hours need to be aligned so that they can communicate with each other. This helps avoiding rework and provides clarity of project.

3.4 Communication

Reduced communication has more effects in case of distributed teams. Most agile practices like test driven development can be educated by providing one-on-one training. Many problems in distributed agile development are related to communication like unable to understand the customer, the system architecture or system design. These have to be solved by participating in discussions or solving the problem manually.

3.5 Knowledge Transition

Knowledge transition is absent in project development, processes of customer support, domain and central product. The developing teams have to set up the overlap times for different time zones so as to achieve the 24 hours and 7 days yield.

3.6 Cultural Differences

Cultural issues can cause misunderstanding between team members. Several recent studies [8, 9, 10, 11, 12] have explored the cultural differences and measures to manage them in distributed teams.

3.7 Lack of Team Cohesion

In case of distributed development, members at distinctive locales are more averse to observe themselves as a major aspect of the same group when contrasted with co-placed members. Absence of togetherness, accompanied by common view of goals, is an issue in that situation. They get worse when we talk about agile development because it focuses on regular collaboration on all phases of the software project.

3.8 People vs. Process Oriented Approaches

In agile development, the process is people oriented and informal methods are used to establish the control whereas distributed development needs the control to be achieved by formal methods.

3.9 Knowledge Management

During the development process the experience of team members, decisions, methods and skills must be gathered through knowledge sharing. This helps the team members to use the experience of the precursor to reduce redundant work and cost. The benefit of global distribution cannot be acquired without effective knowledge or information sharing. Hence knowledge should be managed properly in distributed agile development.

3.10 Language Barriers

The language problem arises when the teams are non-collocated and hence they are not able to understand each other due to their different languages.

3.11 Role of Specialist

Typical software organizations always have people with specialized knowledge like business analyst, testers and user interface specialists etc. Their knowledge is expected to be utilized in the project when needed. Agile methodology does not have any formal mechanism to request such expertise. Even if the specialists are brought, they may face problems similar to a new team member about gaining the understanding of the project requirements [13].

3.12 Developer Fear of Skill Deficiency Exposure

Some developers fear that agile processes can bring forward their deficiencies. Onsite customers, stand-up meetings, use of storyboards and whiteboards bring the shortcoming of developers in front of the whole team because agile methodology involves constant communication and collaboration. In addition, continuous integration and automated testing mean that developers can't hide poor, low-quality code. Exposing the weaknesses of developers can prove counterproductive [14].

3.13 Recruitment Challenges

It is difficult for agile companies to find right people due to lack of agile-specific recruitment policies. There are only few universities or colleges that incorporate agile methods and skills to their programs. Moreover degree programs tend to rely upon either technical or business skills but rarely involve both [14].

4 The Solutions

In the preceding section we discussed many challenges faced by distributed agile development teams. Making a successful appropriated geographically dispersed agile team is to a great extent about balancing the hindrances to communication due to distribution of teams. Actually numerous geographically distributed groups flounder since they attempt to act as if their group is co-spotted and don't successfully address the extra communication troubles put on them. A large number of the communication problems confronted require commitments from the group to enhance and the support of extra practices and instruments.

Below we propose practises and techniques to help organizations overcome the challenges discussed in the previous section.

4.1 Documentation

Good documentation may also lead to collaboration of agile teams. For example, if the use case diagrams with user stories reduces misunderstandings and hence enhances collaboration in teams. Several tools are used for documentation like issue tracker (Jira) and project management tool (Scrum Works) [15].

4.2 Pair Programming

Pair programming can be achieved by using communication tools, show-and-tell hour (every team member demonstrates his or her work to the entire team and receives feedback) and a daily developer scrum (developers meet briefly to collaborate on technical issues or approaches) etc.

4.3 Different Working Hours

The agile team members working at different locations faces some communication challenges due to different time zones. Although regular scrums and overlapping working hours helps to minimize this problem but still delays are encountered in the work. Because sometimes clarification is required or rework has to be done. Sometimes the changes made by one person affects the work of other persons at different location and these changes are not propagated correctly. The following helps:

- *Developer to developer handshakes* means that the development team should communicate all the changes they have made during their working hours that the team members at other locations should be aware of.

- *End of day status notes* means that every team member should share with others what he/she did during the work hours, build status, and any kind of issues that has to be handled or ignored.

4.4 Communication

Instead of setting up meetings at 15 different locations, it is good to set up impromptu meetings using video conferencing. This saves time, expenses and also provides flexibility to attend meetings at any time. The different categories of tools used to achieve effective communication [16] are discussed as follows:

4.4.1 Social networking tools

Nowadays there are number of social networking tools as well as social softwares available online which allow interactions in groups. Some of them are: Live meetings, email and Video Conferencing etc.

4.4.2 Communication tools

Instant Messaging (IM) is used to get a quick attention of a team member for a short query. There are some IM applications in which the conversation can be stored as a permanent record. This is additionally an extraordinary method of knowing whether a fellow team member is in the workplace, in a gathering and not be irritated, or accessible for discussion.

4.4.3 SCM tools

SCM tools are used to track as well as control the modifications in the software. Some of the SCM tools are: Version controlling tools and Repository.

4.4.4 Bug and issue tracking databases

These are the database that records information related to bugs and issues.

4.4.5 Knowledge centres

Knowledge centres include frequently asked questions as well as technical references.

4.4.6 Collaborative development environments

These environments provide tools for development in teams, for example, worksites and project workspaces. Some of these tools are as follows:

- *Visual Studio Team System.* It allows team members to perceive the current state of the project as well as update the tasks of the individual.

- *Scrum for Team System* It puts into practice burndown charts as well as Scrum task boards to aid with tracking and iteration planning. These tools give the distributed teams an experience of a team room.

- *SharePoint.* It is used for sharing data and recording the team decisions. They also include cameras to capture pictures of whiteboards.

- *Dry Erase board technology.* DEBT is not only used to write and erase but also to store what you wrote. DEBT also supports add-ons. Team members can also make a digital copy of board's data which can be used by the later reviewers to track how the final product is achieved.

4.5 Knowledge Transition

It is achieved by using following methodologies [17]:

- *Maintain product/process repository:* Creating a database to help the development teams in tracking the status of the project, reporting the issues and assigning priorities.

- *Focus on well-understood functionality rather than critical new functionality.* Creating an atmosphere in which both the developer and the client get used to the process, application and tools

- *Short cycle but not time-boxed development.* Using short cycle approach, in which 2-3 advancement cycles were permitted to take 2-4 weeks each one, contingent upon the practicality and the setup time required to comprehend the business space.

4.6 Cultural Differences

This can be accomplished by sharing work practices, understanding cultural differences, managing language barriers, rotating team ambassadors, and engendering cultural awareness [18].

4.7 Team Cohesion

This problem can be addressed by building trust. The trust among the team members is very important because of nominal official control. Some practices were utilized to fabricate the trust between the groups, such as the following.

- *Frequent visits by distributed partners.* The regular meetings between the project manager and the customer were organized in three companies.

- *Sponsor Visits.* During the starting phases senior manager visited the development team to finalize contracts and to establish ground rules. These visits established great amount of trust in teams.

- *Build cohesive team culture:* Creating a firm group society by needing that every group was made up of parts that had advanced former working associations with one another and aggregately controlled all the needed ability.

4.8 People vs. Process Oriented

This challenge can be addressed by:

4.8.1 Continuously adjust the process:

Change the practices to- Planning iterations to finalize requirements and develop design and Documenting requirements at different levels of formality.

4.8.2 Verify the trust:

Some of the practices to verify the process and quality of the product are summarized as follow [17]:

- *Distributed QA:* The onshore QA team check the offshore development team for acceptable quality.
- *Supplement informal communication with documentation:* Informal communication should be used accompanied by the documentation of critical artefacts.

4.9 Knowledge Management

Effective knowledge management is achieved by the following four processes in distributed agile development [19]:

- *Knowledge Generation:* It involves formal training, self-learning, customer collaboration, inception and communities of practice.
- *Knowledge codification:* It involves technical representation, wiki and documentation.
- *Knowledge Transfer:* It involves tools, pair programming, on site customer, discussions etc.
- *Knowledge Application:* It sprints, similar context or problem solving.

4.10 Language Barriers

In geographically distributed agile teams, the frequent communication among team members and between client and developer is very important. If they are from different areas with different language then language barriers may arise in the communication. The ways to overcome this problem are discussed below:

- *ESL (English as a Second Language) Course:* This helps in reading, listening, speaking and understanding.
- *Don't assume understanding:* Check and notice if the colleague does not ask any question than it means he/she did not understand it.
- *Praise colleagues for asking questions:* Employees should be encouraged to ask questions so that they understand properly. They should be praised for being honest about misunderstanding and never allowed to feel inadequate and powerless.
- *Speak slowly and clearly:* The native employees should speak slowly and clearly so that non native can easily understand and have time to ask questions.

4.11 Role of Specialist

Specialist knowledge is required irrespective of what software development methodology is being used. For example, an architect may join the project to create the reference implementation and set the technical direction for the project. The agile team members need to have a certain degree of technical understanding and maturity to take on from the architect once the base framework is in place. Some amount of formalism in form of documentation needs to be introduced to record the recommendations and decisions of the specialist [13].

4.12 Developer Fear of Skill Deficiency Exposure

The developers should be provided an environment where they feel safe to expose their weaknesses. This can be achieved by:

- Allowing feedback outside the stand-ups to document any fears, issues, or concerns inappropriate for discussion in open forum.
- Making stand-up meetings voluntary for junior developers.
- Assigning mentors to new staff.
- Pair weaker developers with more experienced developers, giving them joint responsibility for requirements [14].

4.13 Recruitment Challenges

These challenges can be solved by developing recruiting practices for agile methods to hire people and by putting

newly recruited graduate in agile projects to get hands-on-experience [14].

5 Conclusion

This survey is conducted to uncover the challenges faced by geographically dispersed agile teams and the ways to conquer them. The findings in this work will help organizations to adopt the distributed agile development without worrying about its challenges. The organizations can use the proposed techniques to run successful distributed agile projects. B. Ramesh in [17] discussed whether we can implement distributed agile development and also few challenges and their solutions. J. Sutherland in [7], used the scrum approach to distributed teams but still this approach did not address many of the challenges like building trust, documentation, team distribution and task distribution. D. Batra in [20] discussed the grounded theory to accomplish the challenges related to communication and cultural differences. G. Rodriguez [21] used virtual meetings as the source of communication between the team members. There are many other sources of communication as well that can be applied to distributed teams. The purpose of this literature survey is to bring up all the challenges, the proposed solutions as well as their context at one place. So that the organizations can find the whole picture of different challenges in this paper and can apply the suitable approach in distributed teams.

More in depth research can be done on the techniques used to conquer the challenges in the future and hence distributed teams can have multiple ways out for their problems; allowing distributed teams to successfully build a project with minimal obstacles. Table 1 provides the summary of challenges, proposed solutions and the corresponding context.

Table 1: Challenges, proposed solutions and Context

No.	Challenges	Proposed Solutions	Context
A	Documentation	Use Document Management Tools: issue tracker (Jira) and project management tool (Scrum Works).	4 teams practised it-Two of the teams had participants from TelAviv, France and Florida; and the other two teams had participants in TelAviv only
B	Pair Programming	Use of video conferencing tools or replace this with equivalent practices like Show-and-Tell hour or a Daily Developer Scrum.	General case applied to projects that utilize pair programming
C	Different Working Hours	Use of Developer to developer handshakes and end of the day status notes.	Studied by majority of the outsourcers (9 companies) come from North America and the majority of the outsourcees come from Asia (8 companies)
D	Communication	Use of Tools: Live Meetings, E-mail, Video Conferencing, Instant Messaging, Visual Studio Team System, Scrum for Team System, Share Point, Dry Erase Board, etc.	Studied by 3 organizations practising agile
E	Knowledge Transition	Set up overlap time for different time zones to get 24 X 7 yield. Apply knowledge transfer mechanism.	Studied by 3 organizations practising agile
F	Cultural Differences	Engendering cultural awareness, understanding cultural differences, rotating team ambassadors, sharing work practices, and managing language barriers.	18 Agile practitioners from 10 different software organisations in the USA and India
G	Team Cohesion	Maintain team involvement and cohesion.	Studied by 3 organizations practising agile
H	People vs. Process Oriented	Addressed by two groups of practices: Continuously adjust the process and Verify the trust.	Studied by 3 organizations practicing agile
I	Knowledge Management	Use of Knowledge Management Techniques: Knowledge Generation, Knowledge Codification, Knowledge Transfer, Knowledge Application.	45 Agile practitioners from 28 different software companies in the USA, India and Australia.
J	Language Barrier	Speak slowly and clearly, don't assume understanding, praise others for asking questions, sign up for English as a foreign language course etc.	18 Agile practitioners from 10 different software organisations in the USA and India
K	Role of Specialist	Need for a specialist, formal documentation	IT solutions organization based out of India serving customers in US
L	Developer's Fear	Allow Feedback, making stand-ups voluntary, assigning mentors, pairing	Initially focussed on group discussions in 2008 and then, conducted 17 case studies in 2009, using in-depth interviews with senior personnel
M	Recruitment Challenges	Developing recruiting practices, assigning agile projects for experience	Initially focussed on group discussions in 2008 and then, conducted 17 case studies in 2009, using in-depth interviews with senior personnel

Acknowledgment

This work is funded in part by the Kennesaw State University's Office of the Vice President of Research (OVPR) Pilot/Seed Grant, and by the College of Science and Mathematics Interdisciplinary Research Opportunities (IDROP) Program.

References

[1] J. Sutherland, Agile Principles and Values, MSDN Library (http://msdn.microsoft.com/en-us/library/dd997578.aspx)

[2] B. Fitzgerald, G. Hartnett, and K. Conboy. "Customising agile methods to software practices at Intel Shannon"; European Journal of Information Systems, Vol. 15, Issue 2, 200-213, April 2006.

[3] B. Fitzgerald, N. Russo, and T. O'Kane. "Software development method tailoring at Motorola"; Communications of the ACM, Vol. 46, Issue 4, 64-70, April 2003.

[4] J. M. Bass. "Influences on agile practice tailoring in enterprise software development"; In AGILE India, 1-9, Feb 2012.

[5] Cao, K. Mohan, P. Xu, and B. Ramesh. "How extreme does extreme programming have to be? Adapting xp practices to large-scale projects"; Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Jan 2004.

[6] Agility at Scale Survey: http://www.ambysoft.com/surveys/stateOfITUnion201209.html, June-Sept 2012.

[7] J. Sutherland, A. Viktorov, J. Blount, N. Puntikov. "Distributed Scrum: Agile Project Management with Outsourced Development Teams"; 40th Hawaii International Conference on System Science, 274a, Jan 2007.

[8] J. S. Olson and G. M. Olson. "Culture surprises in remote software development teams"; Distributed Development Queue, Vol. 1, Issue 9, 52, December/January 2003-2004.

[9] J. D. Herbsleb and D. Moitra. "Global software development"; IEEE Software, Vol. 18, Issue 2, March/April 2001.

[10] R. Bavani. "Critical success factors in distributed Agile for outsourced product development"; International Conference on Software Engineering, 75-79, Dec 2009.

[11] L. R. Abraham. "Cultural differences in software engineering". In Proceedings of the 2nd India Software Engineering Conference. 95-100. 2009.

[12] S. Krishna, S. Sahay, and G. Walsham. "Managing cross-cultural issues in global software outsourcing"; Communications of the ACM- Human-computer etiquette, Vol. 47, Issue 4, 62-66, April 2004.

[13] U. Banerjee, E. Narasimhan, N. Kanakalata. "Experience of Executing Fixed Price Off-shored Agile Project"; In ISEC'11 Proceedings of the 4th India Software Engineering Conference, 69-75. Feb 2011.

[14] K. Conboy, S. Coyle, X. Wang, M. Pikkarainen, "People over Process: Key Challenges in Agile Development"; IEEE Software, Vol. 28, Issue 4, 48-57, July/August 2011.

[15] H. Smits, "Implementing Scrum in a Distributed Software Development Organization"; Agile Conference, 371-375, Aug 2007.

[16] C. Young, H. Terashima. "How did we Adapt Agile Processes to our Distributed Development"; Agile Conference, 304-309, Aug 2008.

[17] B. Ramesh, L. Cao, K. Mohan, P. Xu. "Can distributed Software Development be Agile?"; Communications of the ACM, Vol. 49, Issue 10, 41-46, Oct 2006.

[18] S. Dorairaj, J. Noble, P. Malik. "Bridging Cultural Differences"; In ISEC'11 Proceedings of the 4th India Software Engineering Conference, 3-10, Feb 2011.

[19] S. Dorairaj, J. Noble, P. Malik. "Knowledge Management in Distributed Agile Software Development"; Agile Conference, 64-73, Aug. 2012.

[20] D. Batra. "Modified agile practices for outsourced software projects"; Communications of the ACM-The Status of the P versus NP Problem, Vol. 52, Issue 9, 143-148, September 2009.

[21] G. Rodriguez, A. Soria, M. Campo. "Supporting Virtual Meeting in Distributed Scrum Teams". In Latin America Transactions, Vol. 10, Issue 6, 2316-2323, Dec 2012.

PM5: One approach to the management of IT projects applied in the Brazilian public sector

Paulo R. M. de Andrade, Adriano B. Albuquerque
University of Fortaleza (UNIFOR)
Graduate program in applied informatics (PPGIA)
Fortaleza, CE, BRA
paulo85br@gmail.com, adrianoba@unifor.br

Otávio F. Frota, José F. da Silva Filho
Water and Sewage Company of Ceara (Cagece)
Department of Information Technology (GETIC)
Fortaleza, CE, BRA
{otavio.frota, fernandes.filho}@cagece.com.br

***Abstract*—The government seeks to continually improve the quality of services in order to achieve its essential mission to better serving the real needs of society. The Project Management presents itself as an essential alternative to provide this gain in quality and effectiveness in the public sector. The very definition of project refers to the idea of planning and executing tasks in a structured manner by qualified personnel with clear and defined, schedule and budget goal known, beyond the previous establishment of controls and indicators for evaluation of results achieved compared as a function those predicted. These privileges make project management a tool in the service of governance, which excels among other principles for effectiveness, economy, efficiency, transparency and accountability. The objective of this study is to point ways for the adoption and improvement of project management information technology (IT) in Brazilian public administration and how to operationalize this deployment. The findings will certainly help to corroborate the idea that the practices of project management is a safe way to the country's development and the effectiveness of government actions.**

Keywords *— projects, government, methodology, agility*

I. INTRODUCTION

The recognition of projects as an organizational principle and specialized management had its first milestone in military matters. The need for project management brought in its wake the creation of tools, techniques, processes and specific vocabulary, character-oriented planning and control activities, and it was replicated in the corporate world [5].

Strictly speaking, project management found its place within organizations and, in a way, within the academy, due to the institutionalization of a successful business practice [17]. They are increasingly using projects to achieve their business goals, but at the same time report that projects fail to meet these goals or only partially meet. Along with the spread of the use of projects, there was an increase in its complexity. This question has been verified in the KPMG survey, with 600 organizations in different countries, which showed an increase of 88% on the complexity of the projects in the IT field [9].

In scenarios like those faced simultaneously by governments at national and state level concern for improving the efficiency and effectiveness in the management of public projects is even more important since it is through the success of these projects that public institutions can achieve the goal of provide quality services to citizens and solve their complex organizational problems, the latter being the main focus of publications in various specialized journals [18].

We have to take into consideration that today organizations use IT to achieve business goals and objectives, organizational efficiency and effectiveness, innovation, growth and competitive advantage. The role of IT has evolved from 'facilitator' to 'innovative' business, however, some authors and executives see it as commodity and not as a competitive advantage due to their ease of being copied. The growing dependence of companies in relation to information and IT suggests that it is worth the managers expend time and attention in the structuring of good governance [21].

The alignment of IT strategy with business strategy is a recurring theme in academic research and some models have been developed to identify the degree, or maturity, in addition to the alignment of practices. Although literature has disagreements about how this alignment is achieved and sustained over time, and on the use of linear metric, the subject stands out among the main concerns of business for more than two decades [4].

At this point, it was noted the need to specify an approach to provide the necessary support to IT project manager in a public company. This approach should cover the peculiarities of IT management, yet encompassing the principles of agile methodologies, such as simplicity. Thinking about it, this paper aims to demonstrate the proposed approach to this case, where we chose to develop a methodology that utilizes best practices of the market, obtained through studies and benchmarking. The remainder of the paper is organized as follows: section 2 describes the related work; Section 3 talks about the reasons of failures in IT projects; section 4 presents the approach; Section 5 shows the use of the approach and the results achieved; Section 6 outlines some conclusions and future work.

II. RELATED WORK

Papke-Shields et al. [13] concluded that there are indications of the relationship between the level of use of practices and the relative success of the project. To better understand the issue of the degree of use of the postulates of the PMBOK Guide, the project success measurement was used. Thus, we sought to identify the degree of adherence to project management practices be correlated with the success.

In previous studies, the central measure of the success of a project is reflected by three aspects: cost, time and scope /

quality. They are called triple constraint (triple constraints or iron triangle)[2]. Over the past decades, the issue of project success has been debated. In addition, there have been many efforts to equip the project manager with the tools and techniques useful to pursue the project management success.

According Cavarec [3], there are other factors that influence during the definition, planning and definition of the cost, time and deliverables of projects. Thus, project managers end up overestimating or underestimating them them. Therefore, the success of the project management is concerned with the capacity of your manager to identify, negotiate, mediate and integrate various project constraints in that work.

According to Andrade and Albuquerque [1], PMI in its annual Benchmarking has posted on its website, the growth rate of success of the projects in Brazil has increased year by year due to the spread of practices and project management techniques. This suggests that the implementation of appropriate project management processes has been recognized as a key factor of success, but which of these good project management practices lead to success has not been explored more precisely.

The study Papke-Shields et al. [13] focuses on this differentiation, as there is consensus that the performance of a project goes beyond these aspects. Cavarec [3] says in his studies that, among other factors, the success of the project depends on the business vision and the point of view of stakeholders. However, it remains the question that if the success of the project management would occur by adherence to the principles of project management guides, such as the PMBOK Guide [16], and these would be influenced by the existence of a Project Management Office (PMO), the company or organization.

III. Why IT Projects In The Public Sector Fail?

In passing a few decades, has seen explosive growth in management of information technology projects, however, this area is still not as mature as other disciplines and areas. This growth is most noticeable in developing countries, where the IT industry is still in its infancy and the sufficient level of maturity has not been reached by most companies. These IT companies in developing countries face the challenges of extreme shortage of qualified and experienced staff, and greater employee turnover. As a result, the IT sector in Brazil is not able to provide adequate support for public and private sector organizations. This is something easy to observe through the annual surveys conducted by the world, and in Brazil, the PMSURVEY, showing data on the management of projects in the country related to the complexity, the success rate among other data [15].

Much has been written about the extent and causes of project failures and numerous studies have discussed a range of recognized risk factors. The most famous, and most cited studies, are The Standish Group International [20], The OASIG Survey [12] e The KPMG Canada Survey [10]. These research works have mainly been carried out in the public and private sector of developed countries.

However, little research has been made to carry out such studies in developing countries. In addition, there is no consolidated data and available on the software project performance in the public sector of developing countries.

After performing a literature review, some factors were identified in common between developing and developed countries, that influence the failure of projects. These factors will be considered as lessons learned in order to be avoided or mitigated in order to achieve project success. These factors are listed below.

- **Changes in government:** when changes in governments, and respectively, the responsible / managers of IT departments, projects already underway end up losing the importance they had, and are often abandoned without completion, due to the new management change priorities before existing.
- **Absence of top management commitment to the project:** it is often the case in public bodies, top management does not give due importance to IT and its projects.
- **Resource Availability:** in public agencies there is always the possibility of non-availability of funds at the appropriate time. The funds often rely on tax revenues or services, or the release of a larger entity.
- **Inexistence of an IT department:** most large public organizations have only a small cell. Medium and small organizations often do not even these small cells, and most importantly, have few technicians.
- **The non-participation of end users in the requirements survey process:** often, the areas requesting IT projects tend to make decisions about system requirements based only on their own experience and knowledge, regardless of knowledge or facilities who will use the system. This kind of attitude causes discomfort to the users, who often have difficulty in assimilating complicated features. For this reason, many systems fall into disuse. This disuse causes the project is not considered a success.
- **Absence of Project Risk Management:** this factor has been highlighted by studies in developed countries as a major cause of failure. This area is almost entirely neglected, because many agencies have a philosophy of "for yesterday", spending more time on rework and fix problems in the project planning and risk management.
- **Resistance to change:** due to lack of computer knowledge and fear of the transparency of its work, end users tend to resist as the implementation and deployment of the systems. For them will be additional work.

IV. PM5 or Project Management 5

During 2011, it was perceived that the Company Cagece had the need for an approach to the management of IT projects more specific, because the current approach based on PMBOK, could not satisfactorily meet all the IT needs and a public company. In 2012 was set up a working group with the

aim of developing a new methodology for the management of these projects. As part of the work, the benchmarking was done through a survey of best practices in project management and portfolio. Through this research, were identified with the companies, the main difficulties existing project management, which are:

- Bureaucracy and lots of processes;
- Lack of standardization of documents;
- Very superficial risk management;
- No prioritization in deliveries;
- Changes are monitored, but the setting;
- Incentive absence;
- Many changes in projects;
- Customer feels abandoned the project;
- Different forms and confusing procurement and contracting provided by law.

With these difficulties, it was decided to mount an approach based on best practices of the PMBOK, the agile development and ITIL, in order to support all IT projects of a public sector company and implement its own project office. ITIL were used concepts of change management and configuration to establish how it would work this integrated control. Of agile methodologies were used concepts dealing deliveries, scope and customer interaction. For other processes, we used the knowledge areas of PMBOK and its concepts, to adapt them to the reality of IT or even the reality of the public sector.

The main objective of the methodology is to assist and simplify project management for the IT area. Thus the method can meet the specific needs of more simplified form. This simplification was achieved by reducing the number of processes methodology, which was previously based on the PMBOK 5th Edition and its forty-seven activities to only nineteen activities, divided into five processes. It is also focused a special attention to how the risks are treated and acquisitions. The following are given information about each process.

A. Process 1 - Integration Management

The Management of Project Integration is the core of project management, and consists of the processes of everyday life with which the project manager has to ensure that all parts of the project work together. It is an ongoing process that the complete manager to ensure that the project proceed from beginning to end, is the daily activity to complete the project work.

The need for integration in project management becomes evident in situations where individual activities interact. For example, a cost estimate needed for a contingency plan involves integration of the planning processes described in more detail in the Project Cost Management processes, project time management and project risk management. When additional risks associated with various staffing alternatives are identified, it is necessary to review one or more of these processes. The project deliverables also need to be integrated into the ongoing operations of the performing organization or the customer organization or the strategic long-term planning, which takes into account future problems and opportunities.

Manage the integration of the project is to ensure that the project components need to work together and is project manager of the paper make that happen. It requires skills in negotiating and managing conflicts of interest. It also requires general management skills, good communication, organization, technical familiarity with the product, etc.

We divided the process into three parts: the development of detailed Project Plan, the implementation of the detailed project design and completion of the project. In addition, the management of integration is also responsible for change control and control of the project configuration.

B. Process 2 – Scope Management

The Scope Management includes activities to ensure that all the necessary work and just enough to make the project successfully and meet the agreed stakeholder requirements are documented. Will ensure the documentation of all functional and non-functional requirements of the project and the final product. This process is connected with the control of what is and is not included in the project. In the project context, the term scope can refer to:

- Product scope: The features and functions that characterize a product, service or result;
- Project Scope: The work that needs to be accomplished to deliver a product, service or result with the specified features and functions.

The activities performed here are essential for the project to have the success expected by stakeholders and therefore requires greater dedication and management work, so that all needs are identified and detailed, focused on the agreed delivery. The other project management processes will catch the information generated in the process to make the confirmation of the work and its validation.

He is also responsible process for ensuring that the client receives exactly what you need, when you need it with the required quality. In this process will be analyzed by the client work packages defined to be done the proper prioritization of deliveries. This prioritization may change during the project because reflect what has more impact to the client what he needs more immediate. With this, will be the project planning, based on prioritized work packages.

The Scope Management plays a key role in the project that is the guarantee of quality, which also will ensure that project deliverables have been successfully completed, following all the specifications and information contained in the project scope. In this process will be determined quality policies, objectives and responsibilities, so that the project meets the needs for which it was undertaken.

This process becomes important because failure to meet the product quality requirements or design can have serious negative consequences for any or all of the project stakeholders. For example:

- Meet customer requirements overloading the project team can result in increased friction between employees, errors and rework.

- Meet the objectives of the project schedule rushing planned quality inspections may result in undetected errors.
- The scope management recognizes the importance of:
- Customer satisfaction: Understand, evaluate, define and manage expectations so that customer requirements are met. For this, a combination of compliance with the requirements is necessary (to ensure that the project produces what it was created to produce) and fitness for use (the product or service must satisfy real needs).
- Prevention instead of inspection: One of the fundamental principles of modern quality management determines that the quality must be planned, designed, and built-in rather than inspected. The cost of preventing mistakes is generally much less than the cost to fix them when they are found by inspection.
- Continuous improvement: The PDCA (plan-do-check-act) is the basis for quality improvement as defined by Shewhart and modified by Deming. In addition, quality improvement initiatives undertaken by the performing organization, such as TQM and Six Sigma should improve the quality of project management and also the design of the product quality.
- Responsibility of management: Success requires the participation of all members of the project team, but remains the responsibility of management to provide the resources necessary for success.

C. *Process 3 – Schedule Management*

The Project Schedule Management is the process responsible for setting the start time and end of the project and each activity and seek to ensure the timely completion of activities and project. The schedule management activities are: specify the project activities, estimate the time and resources of the activities, develop the delivery schedule and manage the activities identified in the schedule: communication, quality, risk, procurement, delivery and changes.

In some projects, particularly the smaller ones, the first three activities have narrowed their limits, being treated with a single activity, and can be run by one person in a relatively short time.

That process interacts with the other processes. Each activity occurs at least once and in one or more phases of the project. This process defines all the steps required to build, update and schedule control. Show tools and techniques to specify the activities and define dependencies.

We have that for every change in the project that impact the design process, will have to go through the process of change and, if necessary, generate a new baseline, with new artifacts impacted. The baseline of a project is a picture of the situation in a moment the project, which will serve as a reference in assessing future results obtained and can address various aspects of the project, such as scope, time and cost. In timeline view, the baseline will enable define whether a project is running or not within the prescribed period, allowing you to define the time to implement corrective actions, fitness and / or communication.

After completion of the agreed schedule between the parties concerned, the commitments of each responsible for tasks should be requested. After the commitment of all, should be asked to approve the project sponsor.

The process activities should cyclically occur throughout the project execution. In charge of defining the project manager should occur daily, fortnightly or monthly meetings to ensure the implementation of all activities of the process. Monitoring meetings should occur throughout the project life cycle.

D. *Process 4 – Resource Management*

Project Resource Management is the process responsible for defining all necessary activities to ensure the resources required for the success of the project by the agreed deadline and cost to be estimated. The schedule management activities are: to analyze resource requirements, define project costs, acquire resources and manage resources (people, hardware, software, information).

The features of a project are all people, consumption of materials and equipment necessary to perform each activity. The types of design features are: labor (professional: analyst, developers, architects, etc.), equipment (hardware and software - computers, servers, systems and applications), materials and supplies (paper, books, printed, etc.). For human resources will be defined a "Plan for Human Resource Management." It is the formal document that describes the procedures necessary to manage all project manpower.

Each activity should take place at least once in the entire project or in one or more phases of the project. This process is strongly linked to the schedule management process. All actions of this process can significantly change what was on schedule.

E. *Process 5 – Risk Management*

The risk term is used to denote the result of the combination of the probability of occurrence of a certain event, random, future and that is independent of human will, and the resulting impact if it occurs. This concept can be even more specific to classify the risk as the probability of occurrence of a particular event that generates loss (downside risk) or profit opportunity (positive risk). The mere fact that an activity exists, opens the possibility of the occurrence of events or combination of them, the consequences are opportunities to gain advantages or threats to success.

The Project risks of Management is the constant process of identifying project risks, analyze exposure and its impacts, plan response, monitor and address the risks as planned. The main point of managing risk is to assess the uncertainty of the future in order to make the best decision possible.

The objective of managing risk is not necessarily eliminate them, but understand them in its fullness, being positive or negative. With the knowledge of the risk, we seek the benefits of opportunities with the positive and minimize the impact of the negative aspects.

A key challenge in risk management is the change of thoughts and actions of the "owners" of risk. Risk management can be defined as: cultural, procedural and structural, relating to realize opportunities while managing to adverse effects. Proper treatment of the risks can determine the success or failure of a project. Should be considered in risk analysis aspects of the project, the resources (human, financial and material), technology, technical, business, and the requirement of influences (internal and external), to minimize the effects of risks on the success of the project to a minimum. It is a set of techniques aimed at minimizing the effects of uncertainties to the project, while minimizing the loss of the organization.

V. The Study Case and Results

We conducted a study of action research, where the researcher may intervene in the project. The study was conducted with the development project of the new company intranet. The scope was similar to the previous version, which was developed using another totally focused approach to the PMBOK. Thus, we could do the analysis before and after the project.

The diagnosis for this action research consists of the following items:

- Problem Description: manage projects more streamlined and efficient way in the public sector;
- Work context: Project to create new intranet with the involvement of 6 managements of Cagece;
- Previous process: Process using only the PMBOK, changes were not handled well, time allocation was not working;
- Theme Search: Applying the proposed approach to prove its efficiency.

During the action research the following actions were taken:

- Explanation how the action research works;
- Training team in the new approach;
- Created the Project Charter and Mental Map;
- Set meetings always on Monday;
- Defined deliveries every 15 days with meeting with the client;
- Improved checklist of delivery;
- Simplified project scope document;
- Simplified the change management process.

Table 1 shows some comparisons between the current project and the previous project.

TABLE I. TABLE 1– ANALYZED FACTORS BETWEEN TWO APPROACHES

Analyzed factors	Results	
	Old approach	New approach
% Of rework in the total scope	16%	5%
% Of occurred risks and unidentified	18% (2 de 11)	0%
% Of total time x planned time	+25%	-2%
% Of customer satisfaction in internal evaluation	78%	90%
% Of planning time spent on documentation	15%	7%
% Of execution time spent on meetings	5%	9%
% Of time spent beyond the planned in acquisitions	10%	0%
Amount of change requests	21	6

From a financial point of view, scope and term, it can be considered that the project was a success and met the expectations of the executive body of the project, using the new approach. Less time was spent on unnecessary activities, more focus was on the project, there were no unforeseen risks and resources were released in the time also ensuring quality of delivery and continuity of operation and team work on other projects.

VI. Conclusion

These results suggest that the proposed approach is a useful methodology for managing IT projects.

The first important observation refers to the results obtained by using Cagece GP methodology. Over the years of use of the approach based solely on pmbok, the finished projects between 10% and 20% more than originally planned time. In the study, the project finished on schedule.

In addition to the reduction in the average development time, there was an increase in the quality of their projects. This finding was made through a window that was already used by Cagece: average assessment per project made by the customer. Previously, customer satisfaction always revolved around three points (on a scale of one to five) and the project using the new approach, the average ultimate satisfaction was 4.5 points.

The interviews and the analysis of industry best practices generated several observations of changes made in the classic method originally deployed in Cagece. These observations were cleared, excluding those related to: organizational changes resulting from the political scene; specific

characteristics of the company, and; changes in the level of maturity.

At the end of the debugging process of observations, the real needs for IT Project Management were raised, and it was generated the approach proposed here in this work.

It is understood that these best practices are tools that would hardly be implemented by those involved in adaptations without their prior knowledge. The exception is the balancing and leveling of human resources in research projects. In fact, the survey is shown in Chapter 3 of this work, the resource imbalance was appointed as one of the shortcomings of the current project management methodology.

As the study of action research of this work dealt specifically with public sector organizations and software development projects, a future study to be done is to include in this study other business segments and other types of projects, aiming to test the validity of the proposed approach. Deeper into the factors that influence the management of IT projects in Brazil is an excellent research subject, allowing compare the experiences of successful international companies.

ACKNOWLEDGMENT

The University of Fortaleza and the Water and Sewage Company of Ceara support this work.

REFERENCES

[1] ANDRADE, P. R. M., e A. B. ALBUQUERQUE. "Escritório de projetos: Características, vantagens e o planejamento de sua implantação no setor público." RBGP. Revista Brasileira de Gerenciamento de Projetos, 17 de Setembro de 2014: 21-26.

[2] ATKINSON, R. "Project management: cost, time and quality, two best guesses and a phenomenon, it's time to accept other success criteria." International Journal of Project Management, 1999: 337-342.

[3] CAVAREC, Y. "Revisiting de Definition of Project Success." PMI Global Congress Proceedings. Vancouver, Canadá: PMI, 2012.

[4] CIBORRA, Claudio U. "Risk, Complexity and ICT." Cheltenham (Edward Elgar), 2007: 23-45.

[5] CRAWFORD, J. K. The Strategic Project Office: A Guide to Improving Organizational Performance. New York: Marcel Dekker Inc, 2002.

[6] EMAM, K. EL, e A. G. KORU. "A Replicated Survey of IT Software Project Failures." IEEE Software, 2008: 84-90.

[7] GEMUENDEN, H. G., e T. LECHLER. "Success Factors of Project Management: The Critical Few." Innovation in Technology Management - The Key to Global Leadership. PICMET '97: Portland International Conference on Management and Technology. Portland: IEEE, 1997. 375-377.

[8] HALL, M., R. HOLT, e D. PURCHASE. "Project sponsor under new public management: lessons from the frontline." International Journal of Project Management, 2003: 495-502.

[9] KPMG. Global IT Project Management. 2015. http://www.kpmg.com (acesso em 28 de Janeiro de 2015).

[10] KPMG. The KPMG Canada Survey. Maio de 2014. http://www.kpmg.com/ (acesso em 16 de Março de 2015)

[11] MAESSCHALCK, J. "The impact of new public management reforms on public servants' ethics: towards a theory." Public Administration 82, nº 2 (2004): 465-489.

[12] OASIG. The OASIG Survey. 1995. http://www.parthenon.uk.com/project-failure-oasig.htm (acesso em 16 de Março de 2015).

[13] PAPKE-SHIELDS, K. E., C. BEISE, e J. QUAN. "Do project managers practice what they preach, and does it matter to project success?" International Journal of Project Management, 2010: 650–662.

[14] PILLAY, S. "A cultural ecology of new public management." Internation Review of Administrative Sciences 74, nº 3 (2008): 373-394.

[15] PM SURVEY.ORG. Report 2014 Brazil. Fevereiro de 2015. http://pmsurvey.org/ (acesso em 29 de Março de 2015).

[16] PMI. A Guide to the Project Management Body of Knowledge: PMBOK(R) Guide. 5ª. São Francisco, CA: Syba - PMI Publishing Division, 2013.

[17] REGO, M. L, e H. A. R. IRIGARAY. "Gerenciamento de Projetos: Existe Produção Científica Brasileira?" Anais do XXXV Encontro ANPAD. Rio de Janeiro, RJ: ANPAD, 2011.

[18] SÖDERLUND, J. "Building theories of project management: past research, questions for the future." International Journal of Project Management, 2004: 183-191.

[19] SRIVANNABOON, S., e D. Z. MILOSEVIC. "A two-way influence between business strategy and project management." International Journal of Project Management, 2006: 493-505.

[20] The Standish Group. Chaos Report. 2014. https://secure.standishgroup.com/reports/reports.php (acesso em 16 de Março de 2015).

[21] WEILL, P., e J. W. ROSS. Governança de TI: Como as empresas de melhor desempenho administram os direitos decisórios de TI em busca de resultados superiores. São Paulo, SP: M. Books, 2006.

[22] WISE, L. R. "Public management reform: competing drivers of change." Public Administration Review 62, nº 5 (2002).

SESSION

TESTING, VERIFICATION, VALIDATION METHODS + SECURITY ANALYSIS, ENERGY EFFICIENT SOFTWARE + SOFTWARE QUALITY ISSUES

Chair(s)

TBA

Modelling the Energy Cost of Application Software for Developers

Fadwa Abdulhalim, Omar Alghamdi, and Kshirasagar Naik
Dept. of Electrical and Comp. Engineering, University of Waterloo, Waterloo, Ontario, Canada, N2L3G1

Abstract— *In this paper, we present a non-exclusive test bench to measure the power consumption of an application running on a server. We provide a modelling procedure and tools to software developers to evaluate energy performance of their applications. A neural network model (NNM) has been trained based on process count information gathered by CollectD and actual real-time power consumption monitored by a TED5000 power meter. By using measurement of an actual system running different workloads, power models for four subsystems (CPU, memory, disk and network interface) on two platforms (two real servers) are developed and validated. Through the use of this modeling procedure, a developer can estimate the system power consumption without the need of using an actual power meter device. Overall, this paper helps the developers to analyze their applications in term of power cost on real servers.*

Keywords: Energy Performance Counter, Application Software Power, Modeling, software developer

1. Introduction

Electrical energy is fundamental to all computer systems [1], [2]. However, it is not ubiquitous, and it is expensive. Those in the business of operating high-volume data centres, servers, and bitcoin mining farms all collectively have an interest in understanding how their systems' resources are utilized. Often times however, fitting vast numbers of components with thermal detection instrumentation is not economically feasible.

Reducing power consumption in computational processes is also important to software developers [3], [4], [5]. Ideally, a tremendous amount of software design goes into considerations that are critical to power efficiencies of computer systems. Sometimes, software is designed by a high-level developer not aware of underlying physical components of the system architecture, which can be exploited. Furthermore, even if a developer is aware, they design software geared towards mass end-user adoption and thus go for cross-compatibility. The challenge for the software designer is to utilize dynamic hardware adaptations. Dynamic hardware adaptations make it possible to reduce power consumption and overall chip temperature by reducing the amount of available performance. However these adaptations generally rely on input from temperature sensors, and due to thermal inertia in microprocessor packaging [6], [7], [8], [9], the detection of temperature changes significantly lag the power events that caused them.

A work-around to dynamically gauge a system's performance is to use performance counters that have been demonstrated to effectively proxy power consumption [10], [11]. Performance counters count the instances of local processor events and calls, and thus eliminate the need for sensors distributed about various parts of the system. However, considerable modelling challenges exist in relating the statistical count information to what is truly happening internally with respect to power consumption. Consequently, there is considerable financial interest in developing accurate models that use the performance counts to cost-effectively provide up-to-date power consumption information. With customization, engineers can target and customize specific aspects of system responses in response to stress tests.

This work aims to provide energy performance evaluation and power consumption estimation of an application running on a server using performance counters. Counter data of various performance indicators will be collected using the *CollectD* tool. Simultaneously during the test, a Power Meter (*TED5000*) will be used to monitor the actual power drawn by the computer server. Furthermore, stress tests are performed to examine power fluctuations in response to the performance counts of four hardware subsystems: CPU, memory, disk, and network interface. We provide a modelling procedure and tools that help to estimate system power consumption without the need of using a power meter device.

The main contribution of the paper is to develop a methodology to estimate the power profile of an application software during its development stage. Intuitively, the power profile of an application software is a sequence of $< t_i, p_i >$, where p_i is the power consumed by the hardware execution platform of the application at the t_i (time). The estimation methodology applies a neural network model, and each execution platform requires a separate estimation model.

The rest of the paper is organized as follows. In section 2, we briefly present the related work and compare our approach with the other approaches. Section 3 presents the system model of our test bench with the modelling methodology. Collecting and simplifying the data process has been explained in Section 4. In Section 5, we present the model and explain how it is used to predict the power consumption. Model validation and tests have been presented in Section 6. Finally, some concluding remarks are provided in Section 7.

2. Related Works

Diverse models, which are created with performance counter information, have been used as a predictor of temperature [12], total system power consumption [9], [11], [13], and subsystem power [10], [14], [15], [16]. Even though those models are simple and fast, they are sensitive to benchmarks chosen for model calibration. Furthermore, those models do not consider full system power consumption. For overall power consumption, each of those models requires one or more measurement phases to determine the contribution of each subsystem. Although all the components vary in their power demands with respect to certain process calls, on average, microprocessors are the largest consumers of power, while other subsystems constituted 40-60 percent of the total power [11].

Must of those models were built using performance counters, and linear regression has been used to train those models [10], [11]. The feasibility of using performance events to model real-time use of power in a computer system has been effectively demonstrated. Performance events, which are readily visible and accessible from the CPU, are highly correlated to power consumption in all the subsystems, including memory, I/O, disc and processor. The authors of the paper [10] produced effective performance counter models that independently measured power for the six main components within a computer: microprocessor, graphics processing unit (GPU), chipset, memory, I/O, and disk. The authors at [17], [18] developed an automated test bench that developers can use to measure the energy cost of the their applications for their various design choices.

Our work is different from those works as follows: (i) our power model of a system is developed through analyzing the various count information from CPU, memory, disk and network interactions; (ii) our test bench estimates the power cost of an application running on server without the need for power sensing hardware; (iii) neural network model has been used to train the power consumption model. In order to properly model system performance via counter-based models, it is necessary to have a methodology to completely represent the power consumption of a system, which is what we do in this work.

3. System Model of Test Bench

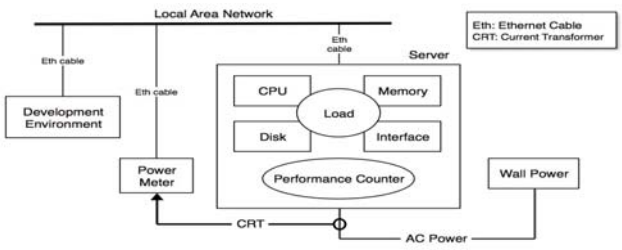


Fig. 1: Test Bench Framework

The test bench in our power modeling process has been shown in Figure 1. The definitions of all the terms are as follows:

Server: A system that runs the software application that the developer is interested in evaluating the energy cost of the application.

Load: It is the software application of which we want to evaluate the energy performance.

Power Meter: An external device that is used for measuring the power drawn by the server. We used TED5000.

Wall Power: This supplies the AC power.

Development Environment: This is a computer equipped with the software tools that are used to analyze and model the data coming from the Server and the Meter.

Our test bench is used to measure the total power cost of a server. To set up the test bench for power measurement, we connected the power meter to the server via a Current Transformers (CRT) and connected the server and the development environment to the same LAN (Local Area Network). Our modelling process involves several steps that have been illustrated in Figure 2.

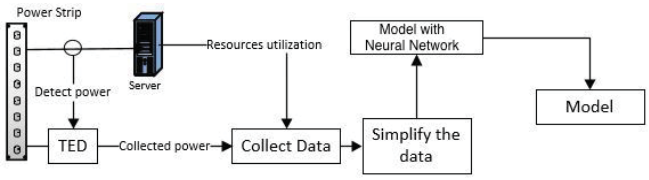


Fig. 2: Modelling Process

The first step is to run the workload and the performance counters (collectD) on the server that is connected to the power meter (TED5000). For the workload, we used a stress tool called *stress*, which is a simple tool used to impose varying amounts of load on the CPU, memory, I/O, and disk. We used it to produce heavy load on selected subsystems (CPU, memory, and disk) in order to drive the basic correlation between their utilization and power consumption. To collect the performance information for each subsystem, we implemented *CollectD*. *CollectD* [19] is an open source UNIX daemon that collects resource performance metrics periodically. It has several plugins to collect data from a specific application or service, and/or to write data to a particular storage medium [20]. We used *cpu, memory, disk* and *interface* plugins, and stored the data as CSV (Comma Separated Values) files on the monitoring station.

The second step is to analyze the collected data to perform modelling. In the development machine, we wrote a Matlab code that helps to simplify the subsystem performance data files. Each subsystem has many metrics collected for it. We reduced the metrics to have only four main metrics, one for each subsystem: CPU, memory, disk, and interface. We will explain the reduction process of the metrics in Section 4.

The final step is to formulate the model based on the performance metrics and the power meter data that have been collected during the load process to predict the power. A Neural Network Fitting (NNF) tool is used to fit the data relating performance metrics to the power variation for the system. By using the four subsystem metrics as input variables and power meter data as a target variable, we performed network modelling that predicts the power consumption.

4. Collecting and Simplifying the Data

In this section, we explain how we collect and simplify the actual power data and the performance metrics for the CPU, memory, disk and interface subsystems.

4.1 Actual Power

A measurement of the actual power is collected by using the energy detective device (TED 5000). *TED500* has been used because it is simple to install and its readings are very accurate [21]. Moreover, this device has a set of Current Transformers (CRT) that clips over the main power cable of a machine. In our test bench, we connect it to the server. The CRT will send the power reading to a Gateway tool and from this tool to the router over an Ethernet cable. The machine that has *CollectD* running on it is connected to the same router. In this way, *CollectD* can take the power reading from TED because they are all on the same local area network (LAN). We added a plugin to the *CollectD* plugins called *Python plugin*. Python plugin collects the power measurements from *TED5000*. The plugin code is as follows:

```
<Plugin python>
    Module path "/usr/lib/collectd"
    Import "ted5000"
           <Module ted5000>
                  Host "129.97.10.85"
                  DkipZeroes "true"
                  Verbose "false"
           </Module>
</Plugin>
```

The *Host Ip address* (e.g, 129.97.10.85) is the TED's getaway address that collects the actual power read and is connected to the router.

4.2 CPU

Regarding CPU (Central Processing Unit), the measurement was CPU usage as a percentage. This measurement was collected via the *CPU* plugin, which measures the fraction of the time spent by the CPU in several states: Idle (idle and without outstanding I/O request), User (executing at the user level), Nice (executing at the user level with nice priority), System (executing at the system level), Wait (idle with an outstanding I/O request), Softer (time the CPU was interrupted by software), Interrupt (time the CPU was interrupted by hardware), and Steal (time spent on other OSs). However, CollectD gives the CPU utilization core by core. For our system that has 8 cores, CollectD will return the utilities for each core separately as CPU1,CPU2,...CPU8.

By applying different loads on the server to observe the CPU states, we found that System and User states have the most performance count. Thus, we calculate the CPU usage as the addition of System and User parameters. We will have CPU Usage = (system + user) and CPU Idle = (idle + interrupt + nice + softer + steal + wait) for each CPU core. Then, we add all the eight *CPU usage* files together to get one *CPU Utilization* file.

It is important to note that *CollectD* collects statistics measured in *jiffies* (units of scheduling), instead of percentage. On most Linux kernels there are 100 *jiffies* to a second, but depending on both internal and external variables, this might not be always true. However, it is a reasonable approximation.

4.3 Memory

For memory resources, we defined the percentage used as the core measurement. This percentage used was calculated using the measurement that was collected via the CollectD *memory* plugin. *Memory* plugin has the four measurement files: Used (used by applications), Buffered (block devices' caches), Cached (used to park file data), and Free (not being used). The Linux kernel minimizes the *free* memory by growing the *cache*, but when an application requires extra memory, *cached* and *buffered* memory are released to give extra memory to applications. Therefore, we considered in the calculation of the percentage used only the memory *Used* measurement file.

4.4 Disk

For hard-disk, we measured the performance statistics using the CollectD *disk* plugin. This plugin gives different measurement files. *Disk-Merged read/write* is for number of read/write operations. *Disk-Octets read/write* is for bytes read/write from/to disk per second. *Disk-Ops read/write* is for read/write operation from/to disk per second. *Disk-time read/write* is for average time an I/O read/write operation took to complete. However, we took the data from *Disk-Octets read/write* files and combined them together in one file called the *disk activities* file. If the server platform has more than one disk, CollectD will return measurement files for each disk. We add all disk activities' files coming from each disk to have one total *disk activities* file.

4.5 Interface

For network interfaces, we used CollectD *interface* plugin, and we chose bytes per second as the unit. The *interface* plugin in *CollectD* collects different measurements as "transfer" or "receive". It provides the rate of error, rate of

bytes, and rate of packets transferred or received. Here we are considering the rate of bytes "transfer/receive" because it is the actual bytes received/transmitted by the network interface. We combined these data files together to have the *interface activities* file. Moreover, utilization is provided by *CollectD* for each ethernet port in the machine.

5. Power Estimation Model

5.1 Workload

The workload is important for developing and tuning the power models. Our workload is chosen based on its apparent utilization of a subsystem. In order to meet the requirement of subsystem utilization, we employ the *stress* tool to produce very high utilization.

The *stress* tool is a simple tool used to impose varying amounts of stress on operating systems. We use it to produce heavy loads on selected subsystems (CPU, memory and disk) in order to drive the basic correlation between their utilization and power consumption.

For a server with eight core CPUs, we fully loaded all of them gradually one by one for one minute per core until we reached 100% CPU load. Also, the memory was loaded gradually to reach 100%. Disk also was loaded with the *stress* tool. For the interface, downloading a large file (about 7 Gigabyte) was the load. In Figure 3, we plot the utilization of the four resources (CPU, memory, disk, and interface) during the stress test. Fig. 3(a) shows the CPU Utilization

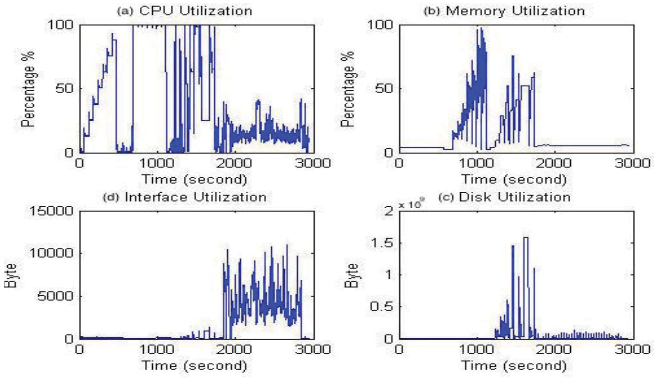


Fig. 3: Resources Utilization During Applying High Load

during stressing of the four subsystems: CPU, memory, disk and interface. From 0-800 seconds, the load was in the CPU. Loading the memory was from 800-1200 seconds. However, the load was in the disk from 1200-2000 seconds, while the interface load was from 2000-2900 seconds. It is clear that the CPU utilization is affected by loading each subsystem. Fig. 3(b) shows how memory utilization has been affected by the load. Fig. 3(c) shows the disk utilization, and Fig. 3(d) shows the interface utilization during the stress load.

5.2 Modelling process

To formulate the power consumption model, we used *Neural Network Fitting* tool. Typically, neural network is trained so that a specific input leads to a particular target output. Here, we used the *resource utilization matrices* as an input and the *actual power watts* as a target output. Both resource utilization metrics and actual power watts are measured and collected per second during the workload.

The resource utilization measurements are compiled into one matrix *RU* with one column for each metric (time, cpu, memory, disk, interface) and a row for each time sample (equation 1). The actual power measurements are saved in another matrix P_a with one column for each metric (time, watt) and a row for each time sample (equation 2).

$$RU = [time, cpu, memory, disk, interface] \quad (1)$$

$$P_a = [time, watt] \quad (2)$$

While we are collecting data from different measurements per second, there are probabilities for missing some data. To observe any missing data, we matched and unified the time of the resource utilizations matrix with the time of the actual power matrix to have one matrix *M* as follows:

$$M = [time_t, power_t, CPU_t, mem_t, disk_t, interf_t] \quad (3)$$

If there is a cell missing data, we will fill it by the average value of the previous cell and the next cell. For example, if we have a missing value (NuN) in CPU column for *t* second (cpu_t), we fill the NuN with the average of the previous value (cpu_{t-1}) and the next value (cpu_{t+1}) as follows:

$$(cpu_t) = \frac{(cpu_{t-1}) + (cpu_{t+1})}{2} \quad (4)$$

After fixing the missing data, we used RU matrix as the input variable and Pa matrix as the target variable based on the following equation:

$$net = f(input, target) \quad (5)$$

where *net* is the model network file, and f is the neural network fitting function.

After training the neural network, we got the estimation of power as an output file. By using the network file (*net*), we can predict the power consumption of a software application running on the same server platform. The performance of the neural network prediction model is shown in Figure 5. The figure presents the actual power, model power (estimated power) and the average error. It is clear that the estimated power is very close to the actual power with Mean Absolute Error (MAE) equal to 14.31 watts.

In Figure 4, we present a generic measurement procedure that can create instance of information model to estimate the power consumption of software application. Developers can follow this simple procedure to get an idea about the power consumption of their software applications during

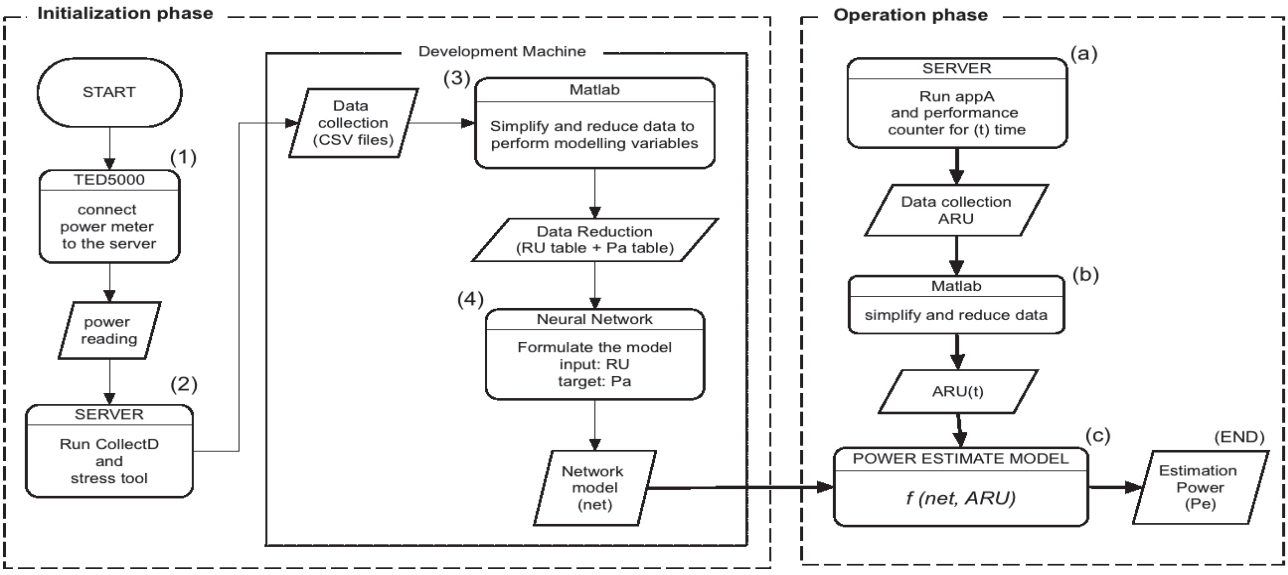


Fig. 4: The Initialization and Operation Phases for The Modelling Process.

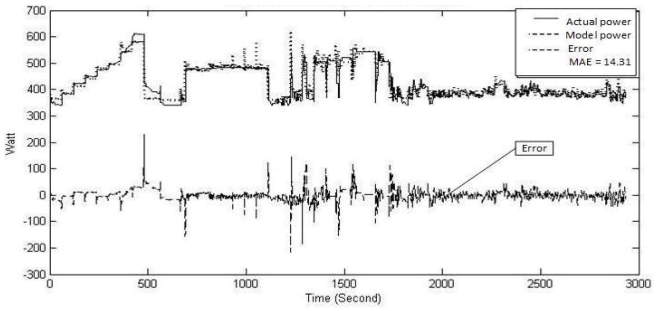


Fig. 5: Estimated power consumption (model power) and measured power consumption (actual power) during the loading of each subsystem to the maximum load.

the development stage. Our measurement procedure involves two main phases.

The first phase is the *Initialization Phase*. This phase needs to be executed only one time for an execution platform, and it includes four steps:

1) Set up the power meter device with the platform for which you need to estimate software application power consumption.
2) Run the workload and the performance counters on the platform which is connected to the power meter.
3) Simplify and reduce the collected performance data to create the modelling variables:
   - Calculate and reduce data to have only four values in resource utilization vectors (*RU*): CPU, Memory, Disk, Interface (Equation 6):

$$RU =< cpu, memory, disk, interface > \quad (6)$$

   - Create Resource Utilization matrix RU_t (Equation 7) and Actual Power matrix Pa_t (Equation 8).

$$RU_t = [cpu_t, memory_t, disk_t, interface_t] \quad (7)$$

$$Pa_t = [watt_t] \quad (8)$$

   - Fix the missing data (NuN) by the average value of the previous cell and the next cell:

$$missing(RU_t) = \frac{(RU_{t-1}) + (RU_{t+1})}{2} \quad (9)$$

4) Formulate the model based on the resource utilization data (RU_t) and the actual power data (Pa_t) to predict the power consumption (Equation 10 and 11).

$$net = f(input, target) \quad (10)$$

$$net = f(RU_t, Pa_t) \quad (11)$$

The second phase is the *Operation Phase*. In this phase, we predict the power consumption of any application running on the same platform that is used in the first phase. This phase is executed once for each case of an application. This phase includes three steps:

(a) Run the software application (*app A*) and *CollectD* simultaneously with the intended scenario for *t* seconds.

(b) Simplify and reduce application resource utilization data *ARU* during the same *t* time.

$$ARU_t = [< CPU_t, Mem_t, Disk_t, Inter_t >] \quad (12)$$

(c) Input *ARU* and training model file *net* to power estimate model function (f_{pem}) to get the estimation power *Pe*.

$$Pe_t = f_{pem}(net, ARU_t) \quad (13)$$

Thus, developers need to create the application resource utilization table (ARU) for the software application that is running on the server by running the application with the performance counter. Then, they will use the modelling network file (net) and ARU table as an input to estimate the application power consumption. Thus, the operation phase is ended by giving the estimation power (*Pe*) data of the application running on the sever.

After a subsystems power consumption model has been trained, it does not require retraining when applied to the same server. So, the developer will train the model once and then apply their applications to predict their power consumption. However, if applying the model to another server platform, retraining is required.

6. Validation

For our experiment, we used two real servers, server 1 and server 2, as described in Table 1 and Table 2, respectively.

Table 1: Description of server 1

Parameter	Real Server (Dell PowerEdge 2950)
Processor	7x Intel Xeon, 3 GHz, 8 cores
Hard Disk	1.7 Tera Bytes SAS
Main Memory	32 GB DIMM
Operating System	Linux (Ubuntu 13.10)
Observable Subsystems	CPU, Memory, Disk and Interface

Table 2: Description of server 2

Parameter	Real Server (HP ProLiant DL385G5)
Processor	Quad-Core AMD Operon^{TM}, 2.3 GHz, 4 core
Hard Disk	55.2 Giga Bytes
Main Memory	15.7 GB DIMM
Operating System	Linux (Ubuntu 14.04 LTS)
Observable Subsystems	CPU, Memory, Disk and Interface

In each server, we applied four different workload scenarios to validate the model. In each scenario, we observed the mean absolute error (*MAE*) of the model. Specifically, we evaluated the Mean Absolute Error of power estimation (MAE_p), the Minimum Error (*MinError*), and the Maximum Error (*MaxError*) which calculated as Equation (14,15).

$$MaxError = \frac{MAE_p}{MaxPower} * 100 \quad (14)$$

$$MinError = \frac{MAE_p}{MinPower} * 100 \quad (15)$$

The *MaxPower* is the highest point the power reached during the workload, and *MinPower* is the lowest point. Moreover, the *Error* was calculated as the difference between the actual power and the model output (the estimation power), Equation (16)

$$Error = ActualPower - ModelPower \quad (16)$$

Also, we calculated the Mean Absolute Error of energy estimation (MAE_e) as Equation 17.

$$MAE_e = \frac{Error}{ActualPower} * 100 \quad (17)$$

For one of these scenarios, we downloaded four movie files at the same time. The total size of the four files was about 7 Gigabytes, and the downloading took around one hour. After collecting and simplifying the data that was collected during the downloading, we applied the model to estimate the power consumption. Figure 6 shows the actual power and the estimation power during the downloading scenario in server 1. The MAE_p was 56.45 watts with minimum rate 13.97% and maximum rate 11.38%, and the MAE_e was 12.64%.

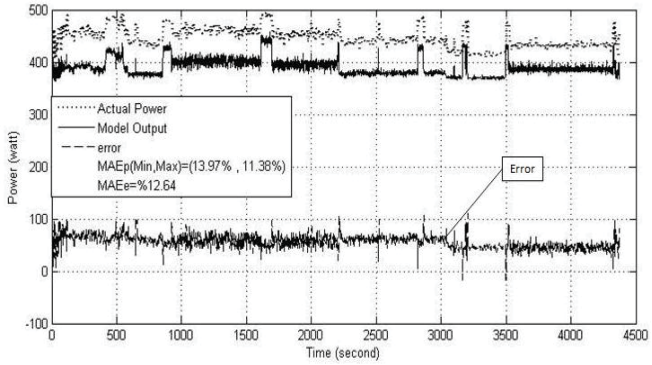


Fig. 6: Estimated power consumption (model power) and measured power consumption (actual power) during downloading 7GB movie files in server1

The second scenario concerned playing a video game for 30 minutes. This game was downloaded to the server. The third scenario concerned playing a high quality video file for about 30 minutes. The video file was also downloaded to the server. The forth scenario concerned plying an online video file on Youtube for about 30 minutes. In each scenario, we observed the mean absolute error of power estimation (MAEp) and the Mean Absolute Error of energy estimation (MAEe). The results are presented in Table (3) and Table (4). The results show that MAEs in server 2 (old system server) are lesser than the MAEs in server 1 (new system server).

Table 3: MAE_p and MAE_e for Various workloads in server 1

Downloading	MAEp (MinError, MaxError)	(13.97% , 11.38%)
	MAEe	12.64%
Gaming	MAEp (MinError, MaxError)	(17.94% , 14.82%)
	MAEe	15.30%
Local Movie	MAEp (MinError, MaxError)	(14.77% , 11.21%)
	MAEe	11.8%
Youtube	MAEp (MinError, MaxError)	(15.59% , 12.52%)
	MAEe	13.15%

Table 4: MAE_p and MAE_e for Various workloads in server 2

Downloading	MAEp (MinError, MaxError) MAEe	(3.66% , 2.78%) 1.52%
Gaming	MAEp (MinError, MaxError) MAEe	(5.07% , 4.00%) 4.17%
Local Movie	MAEp (MinError, MaxError) MAEe	(3.99% , 3.14%) 3.02%
Youtube	MAEp (MinError, MaxError) MAEe	(5.69% , 4.33%) 4.13%

7. Conclusion and Future Work

There is a significant interest in using process count information to model dynamic power consumption. Due to the high correlation of certain system activities, such as CPU utilization, memory accesses and I/O peripherals with power demand, process counts represent a cheap and real-time proxy by which one can estimate the power.

In this paper, feasibility of predicting complete system power consumption using subsystem performance is demonstrated. The framework's infrastructure mainly contains a power meter, server and process count tool. Our test bench captures the power characteristics of a system by correlating a few user-level utilization matrices or hardware performance counters with power consumption during a high load. We performed various loads on two real servers (Dell Power Edge 2950 and HP ProLiant DL385G5) using our test bench. A neural network model (NNM) has been trained based on the process count information gathered by CollectD and the actual real-time power consumption monitored by a TED5000 power meter during applying the workload. Our experiments show that complete system power can be estimated in the new system server with an average MAE of power estimation between 11% to 18% and an average error of energy estimation between 11% to 15%. While in the old system server, the average MAE of power estimation is between 3% to 6% and has an average error of energy estimation between 2% to 5%.

We introduced an easy way to create a model for predicting the power consumption of any application. Moreover, developers can add this kind of testing to their software testing stage to be able to understand the software behaviour from a power consumption point of view. In the future, we will validate our power model methodology in a variety of platforms using more sophisticated models. We aim to reduce the error percentage of the estimated power and increase the accuracy.

References

[1] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[2] K. Naik and D. S. Wei, "Software implementation strategies for power-conscious systems," *Mobile Networks and Applications*, vol. 6, no. 3, pp. 291–305, 2001.

[3] M. Sabharwal, A. Agrawal, and G. Metri, "Enabling green it through energy-aware software," *IT Professional*, vol. 15, no. 1, pp. 19–27, 2013.

[4] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, 2010.

[5] K. Naik, *A survey of software based energy saving methodologies for handheld wireless communication devices*. Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[6] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proc. of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*. ACM, 2000, pp. 37–42.

[7] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunities," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*. IEEE, 2005, pp. 275–280.

[8] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, pp. 93–104.

[9] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 160–171, 2003.

[10] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *Computers, IEEE Transactions on*, vol. 61, no. 4, pp. 563–577, 2012.

[11] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[12] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner, "Event-driven energy accounting for dynamic thermal management," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLPâĂŹ03)*, vol. 22, 2003.

[13] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proc. of the 2001 international symposium on Low power electronics and design*. ACM, 2001, pp. 135–140.

[14] Y. Cho, Y. Kim, S. Park, and N. Chang, "System-level power estimation using an on-chip bus performance monitoring unit," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008, pp. 149–154.

[15] G. Contreras and M. Martonosi, "Power prediction for intel xscale® processors using performance monitoring unit events," in *Low Power Electronics and Design, 2005. ISLPED'05. Proc. of the 2005 International Symposium on*. IEEE, 2005, pp. 221–226.

[16] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 2010, pp. 147–158.

[17] J. Singh, V. Mahinthan, and K. Naik, "Automation of energy performance evaluation of software applications on servers," in *Proc. of SERP*, vol. 14, 2014, pp. 7–13.

[18] J. Singh, K. Naik, and V. Mahinthan, "Impact of developer choices on energy consumption of software on servers," in *Proc. of SCSE'15*, 2015.

[19] A. Datt, A. Goel, and S. C. Gupta, "Comparing infrastructure monitoring with cloudstack compute services for cloud computing systems," in *Databases in Networked Information Systems*. Springer, 2015, pp. 195–212.

[20] G. Da Costa and H. Hlavacs, "Methodology of measurement for energy consumption of applications." in *GRID*, 2010, pp. 290–297.

[21] Z. C. Taysi, M. A. Guvensan, and T. Melodia, "Tinyears: spying on house appliances with audio sensor nodes," in *Proc. of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM, 2010, pp. 31–36.

A Systematic Mapping about Testing of Functional Programs

Alexandre Ponce de Oliveira[1 2], Paulo Sérgio Lopes de Souza[1], Simone R. Senger de Souza[1], Júlio Cezar Estrella[1], Sarita Mazzini Bruschi[1]

1: Universidade de São Paulo, ICMC, São Carlos, SP, Brazil
2: Faculdade de Tecnologia de Lins, Lins, SP, Brazil

Abstract - *Functional languages, like Erlang, Haskell and Scala allow the development of real-time and fault-tolerant parallel programs. In general, these programs are used in critical systems such as telephone switching networks and must provide high quality, reliability and efficiency. In this context, validation, verification and testing activities are necessary and contribute to improving the quality of functional programs. This paper presents a systematic mapping concerning the testing of functional programs, considering also their parallel/concurrent aspects. The paper describes the three stages used during the systematic mapping: planning, execution and presentation of results. The mapping was able to identify only twenty-two relevant studies. In these studies, fourteen considered test models, three used data flow testing, twelve used/proposed testing tools and five considered concurrent/parallel aspects of such programs. The results indicate that there are few researchers working on testing of functional programs and that few studies are concentrated almost exclusively in the Erlang language. The main testing technique found in the papers is the structural one; however, it does not properly consider the software testing methodology already established for the imperative programming. Indeed, the results show gaps in the area of testing of functional programs, even for Erlang, the most considered language by the studies. These gaps are presented and discussed at the end of this paper.*

Keywords: Testing, functional programs, Erlang, testing criteria, test models.

1 Introduction

Nowadays functional programs are an aim of research in universities with distinct examples of research and applications [12]. Parallel and soft-real time features are key aspects related to functional applications, which stimulate the interest for new research.

Functional languages can be used also to build programs utilizing expressions as mathematical functions, avoiding both mutable data and changes in the state of the program that do not depend on the function inputs. The program behavior can be easier to predict when using this paradigm, which motivates research on functional languages. Some examples of functional languages are: Lisp [24], Haskell [37], Scala [37] and Erlang [3] [4].

The functional applications are often critical and failures affect their quality, reliability and efficiency. In this sense, the testing of functional applications is essential to prevent potential failures and to ensure that all features are according to what is expected [6].

Software testing activity aims to find unrevealed defects that are responsible for errors during the execution of programs [25]. A number of studies have been conducted in sequential and concurrent software testing, investigating models, criteria and tools for testing.

Considering the context of concurrent programs, for example, Taylor et al. [38] proposes to apply coverage criteria for concurrent programs. Yang et al. [52] adapts All-Du-path testing criterion for concurrent programs.

Souza et al. proposes structural coverage criteria for C/MPI [33] [35], C/Pthreads [32], BPEL [11] and Java [34]. However, this scenario is not true for the testing of functional programs, since it is not trivial to find studies already published in the context of functional programs (sequential or concurrent).

Functional programs present concurrent aspects and therefore these aspects should be properly explored during the testing activity. In order to contribute to this scenario, it is important to consider state-of-the-art research on functional program testing. We could not find a literature review available in this context, which motivated this work. Considering this scenario, a systematic mapping process was used to collect, guide new research and analyze the papers already published for the testing of functional programs. A systematic mapping identifies, in the literature, what type of studies can be considered in a systematic review, pointing out mainly where those studies have been published and their main results.

A systematic mapping allows a wider view of primary studies, which can reveal the evidences of research [27]. A systematic mapping process is capable to contribute with new research insights in a particular area, providing an initial overview. The systematic review, on the other hand, tries to identify, evaluate and interpret all the available works, relevant for a specific research question [7].

This paper identifies, through a systematic mapping,

studies related to the testing of functional programs, classifying and analyzing relevant papers in this context. The eligible papers were classified under three main features: *a)* work that proposes novel models of testing to functional paradigms; *b)* work that presents testing criteria related to this subject; and *c)* work that presents a software tool to support the testing activity. This classification facilitates the analysis of the selected papers.

The main results indicate that there is little research on testing of functional programs. These studies are focused, almost exclusively, on the Erlang language, using the structural testing technique. However, they do not properly consider the software testing methodologies already established for the imperative programming. It is important to consider this previous research, because the knowledge produced for imperative programming can guide the definition of new approaches for new contexts. Indeed, the results show gaps in the area of testing of functional programs, even for Erlang, the most considered language by the studies.

This paper is structured as follows: Section 2 presents some of the main features of functional languages that make the testing of functional programs different from the testing of imperative programs; Section 3 includes details of the systematic mapping planning; Section 4 presents the execution of the systematic mapping planned; The results are discussed in Section 5 and in Section 6 the main conclusions are drawn.

2 Functional Programs

Functional languages are based on mathematical functions. An important feature of mathematical functions is the use of recursion and conditional expressions to control the order in which pattern matching is evaluated. The variables in functional language are immutable, so once a value is assigned, it cannot be changed; this feature does not generate side effects to the functions [29].

Functional languages have no side effects (or state change), because they define the same value given a same parameter (referential transparency). Functional languages also use higher-order functions; which are functions that receive functions as parameters and can generate a function as a result [43].

A function definition, in functional languages, uses pattern matching to select a guard among different cases and to extract components from complex data structures. Erlang [3] works in that way, combining high level data with sequences of bits, to enable functions of protocol handling.

Concurrency is a fundamental and native concept of some functional languages, such as Erlang. Those languages do not provide threads to share memory, thus each process runs in its own memory space and has its own heap and stack. These processes in Erlang employ the Communicating Sequential Processes (CSP) model [17]. Hoare [17] described how sets of concurrent processes could be used to model applications. Erlang explores this idea in a functional framework and uses asynchronous message passing instead of the synchronous message passing of CSP. Each process has a mailbox to store incoming messages, which are selectively obtained [8].

Some functional applications may run transparently in a distributed environment. In Erlang, a VM (Virtual Machine) instance is called *node*. One or more computers can run multiple nodes independently from the hardware architecture and even operating system. Processes can be created in remote nodes, because processes can be registered in Erlang VM.

Fault Tolerance is a necessary resource for concurrency applications, in this context. Erlang has libraries that support supervisors, worker processes, exception detection and recovery mechanisms. Thus, processes create links to each other to receive notifications as messages. This is used, for example, when a process finishes [23].

3 Systematic Mapping Planning

This systematic mapping was performed according to the process defined by Kitchenham and Charters [18] and Petersen et al. [27]. This process consists of three stages: a) planning – definition of a protocol specifying the plan that the systematic mapping will follow; b) execution – the execution of the protocol planned; and c) presentation of the results [7].

Primarily, our main objective with the systematic mapping was to identify studies that explore the testing of concurrent aspects of functional programs. However, we found few studies in this more restrict context and therefore we decided to make this systematic mapping broader, in order to find a wider range of publications about functional software testing as a whole. Considering this scenario, three research questions were defined and used to conduct the systematic mapping carried out in this paper:

Question 1 (Q1): What aspects related to the testing of functional languages have been identified? Our interest here is to identify the main features in the functional paradigm that make the test activity more complex in this context.

Question 2 (Q2): How is the testing activity of the functional programs conducted? The aim is to find studies that apply testing methodologies and to establish which/how testing criteria are used.

Question 3 (Q3): Are there testing tools for functional programs? Identifying testing tools that support the testing activity is important due to the complexity of the testing activity and the difficulty to apply it manually.

3.1 Search String and Source Selection

The search string was defined as follows: first, the main search keywords were established based on our research questions. We considered terms such as functional language, software testing and testing tools. The languages Erlang, Haskell and Lisp have been inserted in our search string because they are the most used functional languages

for both academic and industrial purposes. However, it must be observed that the string did not restrict the search just for these three languages. Next, a set of relevant synonyms for the search keywords was identified, based on the terminology used in a set of relevant contributions in the area of software testing and functional language. Thus, the main keywords were combined with the chosen synonymous using the Boolean operators AND and OR. The search string used in the systematic mapping is:

[("functional language" or "erlang" or "lisp" or "haskell") AND ("software testing" or "structural testing" or "mutation testing" or "functional testing" or "blackbox" or "whitebox" or "tools" or "test" or "criteria" or "coverage")]

In Table 1 the digital libraries selected to conduct the systematic mapping are presented. These libraries were chosen because they present the most relevant sources in software engineering.

Table 1. Selected Digital Libraries.

Digital Library	Link
ACM	http://dl.acm.org/
IEEE Xplore	http://ieeexplore.ieee.org/Xplore/home.jsp
SCOPUS	http://www.scopus.com/

3.2 Studies Selection

The following inclusion criteria (IC) were defined in order to obtain primary studies that could provide answers to the research questions. It is important to observe that just one valid inclusion criterion is enough to include a primary study in the next step (eliminate primary studies).

IC1: Primary studies presenting testing models for applications written in functional languages;
IC2: Primary studies proposing tools and research for the context of functional language;
IC3: Primary studies applying case studies in the context of functional program testing.

The following exclusion criteria (EC) were defined to eliminate primary studies when they are not related to the research questions:

EC1: Primary studies presenting testing approaches not related to functional languages;
EC2: Primary studies presenting approaches related to hardware testing;
EC3: Primary studies presenting tutorials about software testing or functional languages.

3.3 Data extraction

A form was filled with the extracted data. This form was used to record information obtained from primary studies, as described in Kitchenham and Charters [18]. The data extraction provides information such as: extraction source, title, year and authors. The procedure to extract the data was carried out after the studies. A summary was written for each examined study, in order to facilitate the documentation of the responses for the research questions.

4 Systematic Mapping Execution

The systematic mapping was carried out with the support of the tool StArt (State of the Art through Systematic Review) [30]. Despite its name, related to systematic review, this tool offers facilities to support all the activities of the systematic mappings, including planning, execution and summarization.

The studies were selected in September, 2014 and there were three different stages, as described in the sequence. Initially, 556 studies were retrieved.

In Stage 1, duplicate studies were identified and eliminated (done automatically by the StArt tool). Furthermore, we eliminated non relevant data, such as conference proceedings, abstracts and unavailable papers. After this stage, only 44 studies remained.

In Stage 2, we applied the inclusion and exclusion criteria based on title, abstract and keywords. Moreover, we read the conclusion and the introduction sections of each study in order to apply the inclusion and exclusion criteria. After this stage, only 22 studies remained.

At the final phase (Stage 3), the studies were analyzed completely. In this phase we selected 17 studies. According to our preliminary studies, five other studies were included: [47], [48], [49], [50] and [51]. Such studies were not indexed by digital libraries but were published in local workshops. Thus, 22 studies were selected.

Table 2 shows the number of studies selected at each stage, considering the total studies retrieved from the digital library. All the results of the search procedure were documented and are available[1]. If necessary, the search procedure can be repeated considering, for example, a different period of time.

5 Systematic Mapping Results

This section presents the mapping results, grouping the selected studies according to the research question. The aims of the studies are described as follows.

Q1. What aspects related to the testing of functional languages have been identified?

Widera [51] explains that generating a control flow graph (GFC) for functional programs is more complex than for traditional programs due to the existence of higher-order functions. The difference in the control structures of functional languages in relation to imperative languages also makes the application of the coverage criteria more complex, in the functional context.

[1] http://labes.icmc.usp.br/~alexandre/mapping.pdf, 2014.

Table 2. Number of Studies Selected During the Search Procedure

Digital Library	Return	Stage 1		Stage 2		Stage 3	
		Included	Excluded	Included	Excluded	Included	Excluded
ACM	72	19	53	8	11	6	2
IEEE	171	4	167	1	3	1	-
SCOPUS	315	23	292	15	8	10	5
Total	556	44	512	22	22	17	7

An example of this occurs when higher-order functions can receive and send functions as parameters. This dynamic creation of functions makes the control flow unpredictable and must be considered during the testing activity.

In this same context, Tóth and Bozo [41] cite that the aim of a Data Flow Graph (DFG) is to determine how far a variable definition can reach. This is because variables are immutable in functional languages. In the context of data flow, it is important to analyze a value from its first definition to its last use [50].

Considering the objective of this research question, we identified two main aspects of functions programs that impact the testing activity: higher-order functions and immutable feature of variables. Higher-order functions influence the control flow, which requires a proper analysis of the data flow. The immutable feature of variables brings the necessity of a variable to be copied to another one after its use, so it is important to identify this sequence of copies from its first definition up to its last use. All the studies were related to Erlang language (although, the authors argue that the studies could be extended to consider other functional languages, such as Haskell).

Q2. How is the testing activity of the functional programs conducted?

Tóth and Bozo [41] presented the Semantic Program Graph (SPG), a model to represent the semantic and syntactic information from Erlang programs. SPG is the basis to construct another three graphs: Data Flow Graph (DFG), Control Flow Graph (CFG) and Dependency Graph (DG).

The DG can be used to extract parts of the source code and then identify components that can be parallelized efficiently with inexpensive synchronization. Graphs are integrated in the RefactorErl software, which analyzes the source code and extracts parts of the Erlang code.

Toth et al. [39] investigated the use of SPG during the regression testing aiming to reduce the number of test cases that must be considered to rerun. A behavioral dependency graph is specified and used to represent test cases affected by changes in the program´s behavior, due to modifications. In a similar way, Tóth and Bozo [40] investigate the use of a dependency control graph to support the selection of effective test cases during the regression testing for Erlang programs.

Silva et al. [31] specified a graph called the Erlang Dependency Graph (EDG), which shows the dependencies of data and control in function calls. The authors propose a testing tool, named Slicerl to extract relevant parts of the Erlang program based on the proposed model. Guo et al. [14] defined a model, named Complete Functional Binary Tree (CFBT), which transforms each Erlang function into a tree structure. Each node of the tree corresponds to a predicate of the original function and the objective is to represent all predicates in order to apply coverage criteria based on the CFBT.

Five selected studies, described below in this (Q2) research question, did not consider concurrent aspects of the functional programs although all of them considered Erlang. These studies explore the definition of test models and they do not specify testing criteria.

Three studies discussed in the previous research question (Q1) also contribute to the definition of models and criteria for testing of functional programs. Widera [44] describes a test model to include a subset of Erlang functions and proposes a GFC for this model. This model covers only sequential Erlang programs. In Widera [45] the model is extended to include higher-order functions. Widera [46] complements the model to include concurrency aspects of Erlang programs.

In the context of testing criteria, four studies were retrieved. Widera [47] proposes a set of coverage criteria based on data flow testing adapted to functional programs. These criteria are based on associations of definition and use of variables (du-pair) that is a triple (v, d, u). In this triple, v is a variable, d is a definition of v, u is a use of v and there is a path w from d to u such that v is not redefined on w. Widera [48] introduces the du-chain concept, which is a sequence p1;….; pk of du-pairs, such that the definition of p1 and the use of pk denote the same value. Based on this concept and considering a flow graph G, a set of five testing criteria was defined: a-aware (aliasing aware), s-aware (structure aware), r-aware (result aware), f-aware (freeze aware) and m-aware (message aware).

Tasharofi et al. [36] presents a scalable and automatic approach to test non-deterministic behavior of actor-based Scala programs [1]. This approach uses three schedule coverage criteria for actor programs, an algorithm to generate feasible schedules to increase the coverage and a technique for deterministic execution. Le et al. [19] presents new mutation operators for functional constructs and also describes MuCheck, a mutation testing tool for Haskell programs.

To summarize, we observed contributions that explore mainly the structural testing for functional languages. These papers present propositions to represent and to extract relevant information for testing of functional programs.

Q3. Are there testing tools for functional programs? Widera [49] considers data flow tracing of Erlang codes and describes the properties and implementation of an interpreter prototype for GFC. The interpreter instruments the source code with the aim to evaluate parts of the GFC that are covered by the test cases. The study does not evaluate the coverage criteria; it only makes a comparison of the runtime of small code examples with and without the interpreter.

Nagy and Vig [26] present a survey about the main testing tools used by developers of Erlang systems. The survey is focused on model-based testing and Test-Driven Development (TDD). The tools mentioned by the developers were Dialyzer, EUnit, Wrangler and RefactorErl and QuickCheck, which was proposed by Claessen and Hughes [10]. The survey specifies that the tools used by the developers do not present information about the coverage of test cases and that it is also difficult to know what was really tested into the program. These aspects encourage the improvement of tools available for concurrent functional programs testing.

Christakis and Sagonas [9] present a technique for detecting errors related to message passing in Erlang, considering the dynamic process of creation and communication based on asynchronous message passing. Static analysis is used to build a communication graph from a Dialyzer tool. This graph is traversed during the testing activity, to obtain data about the message passing coverage.

Arts et al. [5] presented a testing tool called Quviq QuickCheck to analyze properties in Erlang programs. This tool uses a model to represent data type information from specification and during the testing; it can be evaluated whether the data types of the program meet its specification.

Wrangler and RefactorErl tools aim to support the refactoring of Erlang programs, the aim of which is to detect a similar code. Taking this into account, Li and Thompson [20] used the Quviq QuickCheck testing tool to automate the refactoring performed by the Wrangler tool. Li and Thompson [39] and [41] proposed a technique to detect syntactically identical codes, which was developed and integrated into the Wrangler testing tool. The authors used both syntactic analysis and code decomposition to remove duplicated code and thus reduce code maintenance.

Gotovos et al. [13] developed the Concuerror testing tool to assist the TDD process. This tool uses test sets to detect errors related to concurrency, such as deadlocks and race conditions in Erlang programs.

Therefore, six studies [9], [5], [20], [21], [22] and [13] are related to model testing, refactoring and TDD. Two studies [9] and [13] explore concurrency aspects.

5.1 Other Results

Figure 1 shows the number of selected studies by year. The result of the mapping showed studies only from the last 11 years, while 2011 had the highest score with four selected studies.

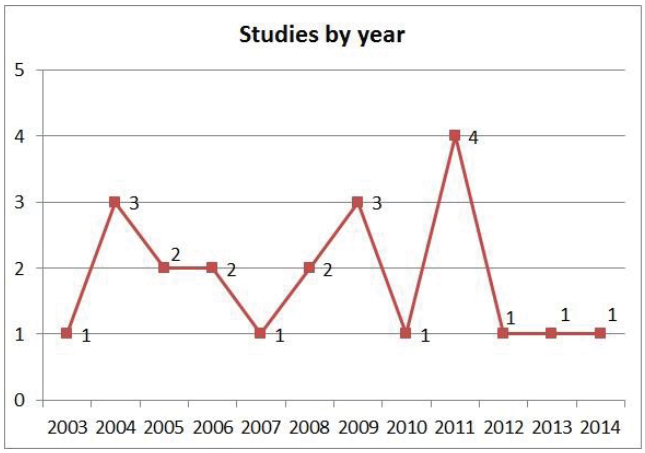


Figure 1. Numbers of studies by year.

Figure 2 groups studies by country, considering the authors´ affiliation. The results show that the University of Hagen in Germany has 8 studies, i.e., 36% of the selected studies. An important feature of these studies is that only four were conducted in partnership with universities in different countries. Two studies were conducted by universities in Greece and Sweden, one study was carried out by universities in Sweden and Spain and one study between universities in the USA and Switzerland.

Figure 3 shows the percentage of selected studies by research question. According to the result, 50% of the studies are related to Q2, which refers to a testing specification models and testing criteria. Q3 is related to testing tools, and 27% of the studies are in this context. Only 9% of the studies specify the challenges of testing activity for functional languages (Q1). Finally, 9% of the studies are in the context of Q1 and Q3 together and 50% of the studies between Q2 and Q3.

6 Concluding Remarks

A systematic mapping conducted to find studies on software testing for functional languages was presented in this paper. These studies provide an overview for the testing of functional languages, revealing the state of the art in terms of knowledge production in this area. These studies point out new research insights and can be used to guide further contributions in this context.

Some studies ([36], [46], [47], [48] and [50]) present the definition of data flow testing for functional programs in Erlang, exploring the definition-use of variables. In this group, five studies ([9], [13], [36], [46] and [48]) investigated the concurrency and parallel aspects existent in functional programs.

The selected studies proposing testing tools for functional programs, consider mainly structural aspects of such programs.

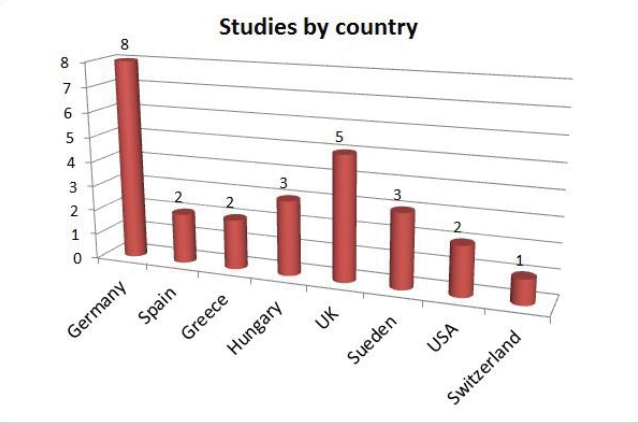


Figure 2. Numbers of studies by country.

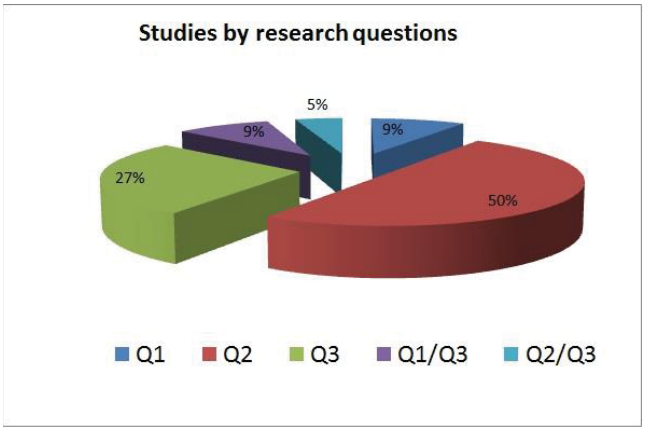


Figure 3. Numbers of studies by research questions.

However, in general, these tools do not apply properly the testing techniques; they do not explore the testing process, such as: generation of test cases and testing activity evaluation.

In summary, 63% of the studies present test models for Erlang programs; 45% of the studies applied a case study to evaluate a testing tool; 18% of all papers define testing criteria exploring sequential aspects of the programs and 9% investigate concurrent aspects of the programs to define testing criteria.

Furthermore, this mapping contributed to indicate lack of research exploring how to derive tests from functional programs and how to extract relevant information from these programs, in order to guide the testing activity. Also, there is a lack of experimental studies to analyze tools and testing criteria.

These results indicate a gap in research related to coverage testing applied to functional programs, mainly related to concurrent aspects of these programs. Considering this gap, we are investigating the definition of structural testing, exploring the same aspects in Souza et al. [33] in this context. We intend to define the coverage testing able to explore intrinsic aspects of this program, for instance: synchronization, communication, parallelism and concurrency considering message passing and other language features such as: higher order functions and functions call.

References

[1] Agha, G. Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge, USA, 1986.

[2] Almasi, G.; Gottlieb, A. Highly parallel computing. The Benjamin/Cummings series in computer science and engineering. Benjamin/Cummings Pub. Co., 1994.

[3] Armstrong, J., Virding, R., Wikström, C., and Williams, M. Concurrent Programming in Erlang. Prentice Hall Europe, Herfordshire, Great Britain, second edition, 1996.

[4] Armstrong, J. Concurrency Oriented Programming in Erlang. Invited talk, FFG. 2003.

[5] Arts, T.; Castro, L.M.; Hughes, J. Testing Erlang Data Types with Quviq QuickCheck. In: Proceedings of the ACM SIGPLAN Workshop on Erlang, ACM Press , 2008.

[6] Balakrishnan, A. and Anand, N. (2009). Development of an automated testing software for real time systems. In Industrial and Information Systems (ICIIS), 2009 International Conference on, pages 193 - 198.

[7] Biolchini, J.C.A.; Mian, P. G.; Natali, A. C. C.; Conte, T.U.; Travassos, G. H. Scientific research ontology to support systematic review in software engineering. Advanced Engineering Informatics, p.133-151, 2007.

[8] Cesarini, F. and Thompson, S. Erlang Programming - A Concurrent Approach to Software Development. O'Reilly Media, 2009. 496p.

[9] Christakis, M.; Sagonas, K. Detection of asynchronous message passing errors using static analysis. In: Proceedings of the 13th international conference on Practical aspects of declarative languages, PADL'11, p.5-18, Austin, USA, January 24-25, 2011.

[10] Claessen, K.; Hughes, J. QuickCheck: a lightweight tool for random testing of Haskell programs, Proceedings of the fifth ACM SIGPLAN international conference on Functional programming, p.268-279, September 2000.

[11] Endo, A. T.; Simão, A. S.; Souza, S. R. S.; Souza, P. S. L. Web services composition testing: A strategy based on structural testing of parallel programs. In: TaicPart: Testing Academic & Industrial Conference - Practice and Research Techniques, Windsor, 2008, pp. 3–12.

[12] Erlang FAQ. Who uses Erlang for product development? http://www.erlang.org/faq/introduction.html, 2014.

[13] Gotovos, A.; Christakis, M.; Sagonas, K. Test-driven development of concurrent programs using concuerror. In Proceedings of the 10th ACM SIGPLAN workshop on Erlang (Erlang '11). ACM, New York, USA, 2011.

[14] Guo, Q.; Derrick, J.; Walkinshaw, N. Applying Testability Transformations to Achieve Structural Coverage of Erlang Programs. In Proceedings of the 21st International Conference on Testing of Software and Communication Systems and 9th International Workshop FATES, Eindhoven, Netherlands, November 2-4, 2009.

[15] Grama, A; Gupta, A; Karypis, G; Kumar, V. Introduction to Parallel Computing. 2nd Ed. Addison Wesley, 2003.

[16] Hansen, M. R.; Rischel, H. Functional Programming Using F#. Cambridge University Press, 2013.

[17] Hoare, C.A.R. Communicating Sequential Processes. Prentice Hall, Upper Saddle River, NJ, 1985.

[18] Kitchenham, B.; Charters, S. Guidelines for performing systematic literature reviews in software engineering. Technical Report. EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[19] Le, D.; Alipour, M. A.; Gopinath,R.; Groce, A. MuCheck: an extensible tool for mutation testing of haskell programs. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014). ACM, New York, NY, USA, p. 429-432. 2014.

[20] Li, H; Thompson, S. Testing Erlang Refactorings with QuickCheck. In the 19th International Symposium on Implementation and Application of Functional Languages, IFL 2007, LNCS, pages 182-196, Freiburg, Germany.

[21] Li, H; Thompson, S. Clone detection and removal for Erlang/OTP within a refactoring environment. In Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation (PEPM '09). ACM, New York, USA. 2009.

[22] Li, H; Thompson, S. Incremental clone detection and elimination for erlang programs. In Proceedings of the 14th international conference on Fundamental approaches to software engineering: part of the joint European conferences on theory and practice of software (FASE'11/ETAPS'11). Springer-Verlag, Berlin, Heidelberg, 2011.

[23] Logan, M., Merritt, E., and Carlsson, R. Erlang and OTP in Action. Manning Publications. 2010.

[24] McCarthy, John; Abrahams, Paul W.; Edwards, Daniel J.; Hart, Timothy P.; Levin, Michael I. Lisp 1.5 Programmer´s Manual. Cambridge, Massachusetts: The MIT Press, 1962.

[25] Myers, G. J. The Art of Software Testing. 2 ed. John Wiley & Sons, 2004.

[26] Nagy, T., Víg, A.N. Erlang testing and tools survey. Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, September 27-27, 2008, Victoria, BC, Canada.

[27] Petersen, K.; Feldt, R.; Mujtaba, S. and Mattsson, M. Systematic mapping studies in software engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08), 2008. British Computer Society, Swinton, UK, 68-77.

[28] Rauber, T.; Rünger, G. Parallel programming: for multicore and cluster systems. Springer, 2010.

[29] Sebesta, R. W. Concepts of Programming Languages. 10. ed. Pearson. 2012.

[30] StArt. State of the Art through Systematic Review. http://lapes.dc.ufscar.br/tools/start_tool, 2012.

[31] Silva J.; Tamarit, S; Tomás, C. System dependence graphs in sequential erlang. In Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering (FASE'12). p.486-500, Tallinn, Estonia, Springer-Verlag, 2012.

[32] Sarmanho, F.; Souza, P. S. L.; Souza, S. R.; Simao, A. S. Structural testing for semaphore-based multithread programs. In: ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I, Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–346.

[33] Souza, S. R. S.; Vergilio, S. R.; Souza, P. S. L.; Simão, A. S.; Hausen, A. C. Structural testing criteria for message-passing parallel programs. Concurrency and Computation: Practice and Experience, p. 1893–1916, 2008.

[34] Souza, P. S. L.; Souza, S. R. S.; Rocha, M. G.; Prado, R. R.; Batista, R. N. Data flow testing in concurrent programs with message-passing and shared-memory paradigms. In: ICCS - International Conference on Computational Science, Barcelona, Espanha, 2013b, pp. 149–158.

[35] Souza, P. S. L.; Souza, S. R. S.; Zaluska, E. Structural testing for message-passing concurrent programs: an-extended test model. Concurrency and Computation, v. 26, n. 1, pp. 21–50, 2014.

[36] Tasharofi, S.; Pradel, M.; Lin, Y. and Johnson, R. Bita: Coverage-guided, automatic testing of actor programs. In 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), 2013.

[37] Tate, Bruce A. Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages. Pragmatic Bookshelf, 2010.

[38] Taylor, R. N.; Levine, D. L. and Kelly. C. D. Structural testing of concurrent programs. IEEE Tr Softw Eng, 18(3):206–215, 1992.

[39] Tóth, M.; Bozó, I.; Horváth, Z.; Lövei,L.; Tejfel, M.; Kozsik, T. Impact Analysis of Erlang Programs Using Behaviour Dependency Graphs. Proceedings of the 3th Conference on Central European Functional Programming School, p.372-390, Komarno, Slovakia, May 25-30, 2009.

[40] Tóth, M. and Bozó I. Building dependency graph for slicing erlang programs. Conference of PhD Students in Computer Science, Periodica polytechnica, 2010.

[41] Tóth, M.; Bozó, I. Static analysis of complex software systems implemented in erlang, Proceedings of the 4th Conference on Central European Functional Programming School. Budapest, Hungary, June 14-24, 2011.

[42] Trobec, R.; Vajteršic, M.; Zinterhof, P. Parallel computing: Numerics, applications, and trends. Parallel Computing: Numerics, Applications, and Trends. Springer, 2009.

[43] Watt, D. A. Programming Languages: Concepts and Paradigms. Prentice Hall International Series in Conputer Science, 1990.

[44] Widera, M. Flow graphs for testing sequential Erlang programs. In Proceedings of the 3rd ACM SIGPLAN Erlang Workshop. ACM Press, 2004.

[45] Widera, M. Towards flow graph directed testing of functional programs. In Draft Proceedings of the 15th International Workshop on the Implementation of Functional Languages, IFL, 2003.

[46] Widera, M. Concurrent Erlang flow graphs. In Proceedings of the Erlang/OTP User Conference 2005, Stockholm, 2005.

[47] Widera, M. Data flow coverage for testing Erlang programs. In Marko van Eekelen, editor, Proceedings of the Sixth Symposium on Trends in Functional Programming (TFP'05), September 2005.

[48] Widera, M. Data flow considerations for source code directed testing of functional programs. In H.-W. Loidl, editor, Draft Proceedings of the Fifth Symposium on Trends in Functional Programming, Nov. 2004.

[49] Widera, M. Flow graph interpretation for source code directed testing of functional programs. In Implementation an Application of Functional Languages, 16th International Workshop, IFL'04. Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universit□at zu Kiel, 2004.

[50] Widera, M. Adapting structural testing to functional programming. In International Conference on Software Engineering Research and Practice (SERP 06), 86-92. CSREA Press, 2006.

[51] Widera, M. Why Testing Matters in Functional Programming. 7th Symposium on Trends in Functional Programming, University of Nottingham, TFP, 2006.

[52] Yang, C.-S. D.; Souter, A. L. and Pollock, L. L. All-du-path coverage for parallel programs. In ISSTA, pages 153–162, 1998.

Cloud-ODC: Defect Classification and Analysis for the Cloud

M. Alannsary[1], **and J. Tian**[1 2]
[1]Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, USA
[2]Northwestern Polytechnical University, Xi'an, China

Abstract— *One of the major contributors to software quality improvement is defect analysis. Current work in this area does not consider multi-tenancy or isolation, which makes it inappropriate to implement on a SaaS (Software as a Service). We propose a defect analysis framework for a SaaS running in the Cloud. The proposed framework is inspired by the ODC (Orthogonal Defect Classification) model, and considers the special characteristics of the SaaS. The framework is composed of three phases: data source analysis phase, classification phase, and results analysis phase. Entries in the web server log file are analyzed to identify errors based on the values of the "Protocol status" field, then observed errors are classified according to six defect attributes specifically tailored to the Cloud environment. One-way and two-way analysis were implemented successfully and resulted in better insight on the observed defects and their resolution. A case study is presented to demonstrate the viability of the new approach.*

Keywords: Cloud, SaaS, SaaS Quality, ODC, and SaaS Defect analysis.

1. Introduction

In defect analysis, we analyze defects observed in a software either during development stages or after release to improve the quality of the software by resolving these defects and eliminating other potential defects that share some common characteristics. There has been several ways and models developed to analyze defects, one of which is through defect classification and related analysis. One of the well-known methods of defect analysis is the ODC (Orthogonal Defect Classification) concept, which has done a good job for in-process defect feedback to the development team [1]. Such feedback is essential for defect elimination and resolution.

Cloud computing is a new technology that has gained a lot of attention. The Cloud is built on the service concept, where most of the computing resources are provided as a service [2], [3]. SaaS (Software as a Service) is using the Cloud to deliver software to users. SaaS is considered a "strategic tool to compete in the market" [4]. SaaS applications have the advantage of multi-tenancy [5], which is serving multiple tenants (with each tenant having one or more users) at the same time using a centralized version of the source code. Adopting any of the multi-tenancy models will enable an application to be considered as a SaaS. The only difference between traditional software and SaaS is fulfilling the multi-tenancy design of the SaaS and assuring isolation in the software and database between tenants [6]. The process of developing software has not changed to the extent that requires a new software development life cycle model for SaaS.

One of the challenges that face SaaS providers is to comply with the QoS (Quality of Service) levels promised in the SLA (Service-Level Agreement). Currently defect analysis for SaaS is in an early stage. There has not been a clear defect analysis model designed specifically for SaaS, despite the differences between SaaS and traditional software. We propose a new method called Cloud-ODC for defect analysis. Cloud-ODC will help eliminate and resolve defects, and on the long run will improve the reliability and overall quality of the SaaS. The new method will be based on adopting the ODC concept to a SaaS running in the Cloud.

The remainder of this paper is structured as follows: The next section presents related work. Section 3 presents our methodology of SaaS defect analysis by adopting the ODC concept. Section 4 presents a case study conducted using a multi-tenant SaaS deployed on Amazon Cloud using AWS (Amazon Web Services). Section 5 presents analysis of the case study results and discussion. Section 6 contains the conclusion and prospectives.

2. Related Work

Software quality is the discipline of software engineering that studies, analyzes, measures, and improves the quality aspects of a software product. Generally speaking, a software product is said to be with high quality if it has none or a small amount of problems that are caused by the software itself and have limited damage to its users [7].

Activities to eliminate injected errors or defects from a system usually consumes more than one quarter or more of its budget. Defects that are bound to requirements, specification, and high level design are damaging and expensive due to the accompanying chain-effect [8]. Thus it is wise to prevent the injection of these errors or defects, commonly by analyzing and eliminating their causes. Such root cause analysis is usually based on previous versions of the software or from similar software products from the same development company or other similar companies. In addition, it is possible to study defect distribution and the logical links between the faults and errors, to predict the infield reliability

of the software through statistical defect models. ODC is one of the well-known and documented methods of defect analysis. It plays a major role in bridging the gap between statistical defect models and causal analysis.

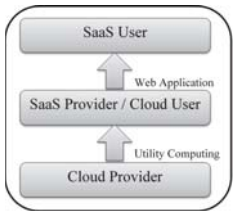


Fig. 1: SaaS configuration in the Cloud [2]

SaaS quality is the process of assuring the quality of a SaaS running in the Cloud. Since SaaS has new features that differentiates it from traditional software, certain modifications to the models and methods used to assure the quality of traditional software are needed to accommodate the new features. SaaS in the Cloud is built using three layers: the Cloud provider, the SaaS provider /Cloud user, and the SaaS user [2] as illustrated in Figure 1. In SaaS we use the Cloud technology to deliver software to tenants and their users. Software development companies need to adopt the SaaS paradigm in order to compete in the market. Xu et al. [9] presented a combined model for defect prediction, measurement, and management. The proposed model is mainly designed for software that is based on SOA (Service Oriented Architecture).

Chillarege et al. [1] proposed ODC (Orthogonal Defect Classification) as a "concept that enables in-process feedback to developers by extracting signature on the development process from defects". In general, ODC is a method of classifying defects to know which stage of the software development process that needs attention. Defects are classified across the following dimensions: defect type, trigger, source, severity, impact, and phase found.

Ma and Tian [10] developed a web error classification and analysis method by adopting ODC to the web environment. They selected attributes that are related to the web environment using web access logs to act as the defect types in ODC. This technique enabled the identification of problematic areas and provided feedback to the web applications development team.

Since there are differences between traditional and web software on one hand and SaaS on the other, specifically in the way the software is structured, hosted, and delivered to its intended customers, such as: multi-tenancy, user licensing, and isolation. New or adopted defect analysis techniques are required for SaaS individually. These techniques must attend to and consider the issues of SaaS defect analysis such as: the constant introduction of new features, continuous update to the SaaS itself, and the capability of selection that tenants have which gives them the option of renting some services and not the SaaS solution as a whole.

3. A New Method

Defect analysis has been successfully implemented on traditional and web software. However, defect analysis for SaaS is currently in an early stage. As mentioned above, there has not been a clear defect analysis model designed specifically for SaaS, to address the new characteristics of SaaS that were not available in traditional and web software.

Our proposed framework is inspired by the ODC model, while considering the special characteristics of the SaaS. It is composed of three phases: the data source analysis phase, the classification phase, and the results analysis phase. In the first phase, contents of the data source (the web server log) are analyzed to locate defects. In the second phase, observed defects are classified by mapping them to known Cloud-ODC attributes. In the third phase, classification results are analyzed either via one-way or two-way analysis. The proposed Cloud-ODC framework is depicted in Figure 2.

3.1 Attributes

In order to benefit from the classification concept of the ODC model in SaaS development projects, special steps need to be taken to accommodate the characteristics of SaaS.

First of all, the list of attributes needs to be modified, either by adding new attributes or omitting currently used ones. The "Phase found" attribute will be omitted, this is due to the fact that all defects are observed during the run of the SaaS and it is difficult to link them to a specific development phase. In addition, the "Layer affected" attribute will be introduced to allow classifying the defects based on its effect on the cloud layers. Table 1 lists the ODC attributes or in other words the Cloud-ODC attributes, where newly introduced attributes are in **bold**, and partially modified attributes are in *italic*.

Second, two more defect types are going to be added to the defect types suggested in the ODC model. The first defect type is for classifying defects based on tenant isolation. Serving multiple tenants at the same time using the same source code file and/or database requires isolating each tenant from other tenants. Thus failing to keep the functionalities, services, or data of a tenant isolated from other tenants is considered a defect. Defects that arise from not complying with tenant isolation affect the QoS of a SaaS, and for that reason it is a necessity to classify them in a new defect type. Therefore, the first new defect type that will be introduced is called: "Isolation".

The second defect type added is related to defects that were caused or influenced by either the IaaS or the PaaS layers. One could argue that failures caused by these layers could be classified as "Interface" defects in the ODC model. This is true except that our goal is to classify the defects for better resolution and elimination. Differentiating between failures caused by either IaaS or PaaS and other

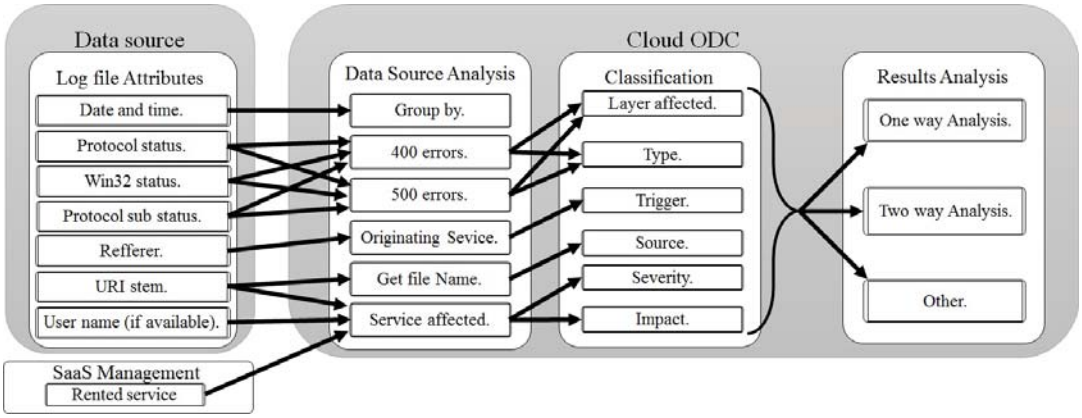


Fig. 2: Cloud-ODC framework.

Table 1: Cloud-ODC attributes.

ODC attribute
Layer affected.
Defect type.
Trigger.
Source.
Severity.
Impact.

SaaS components, models, or device drivers enables us to reconfigure or replace either or both IaaS and PaaS so the SaaS can be fully configured for both.

Moreover, the number of defects caused by the IaaS or PaaS after the SaaS goes live are expected to be fewer than the ones that are discovered during development. This is due to the fact that IaaS and PaaS are the same while developing the SaaS or running it live. Thus adding a new type to the ODC model that is focused on IaaS and PaaS will allow the development team to better reconfigure or replace either IaaS or PaaS early in the development life-cycle. Therefore, the second new defect type that will be introduced is called: "IaaS/PaaS".

It is our intuition that adding the suggested attributes and defect types will allow SaaS providers to better classify defects in their applications, which will eventually provide in-process feedback that will aid in resolving, eliminating, or managing defects to reach the desired level of QoS. The original defect types in addition to the newly introduced ones are listed in Table 2. The newly introduced attributes are in *italic*.

3.2 Data Source

In the proposed approach there are several resources that could be considered as data source candidates by the SaaS development team to discover defects. Such as the bug report, the application log file, the security log file, the user session, the web server log, and other server logs. However, after experimenting with these resources, it was clear that some are not beneficial and do not add value to the new proposed approach. Following is a description of the results we observed after experimenting with some of these resource:

Table 2: Original and new defect types - newly added attributes are in *italic*.

Version	Defect type	Description
Original	Function.	Design change.
	Interface.	Error interacting with other components, Models, or device drivers.
	Checking.	Program logic that fails to validate data and values.
	Assignment.	Control blocks, data structure.
	Timing/serialization.	Corrected by improved management of shared and real time resources.
	Build/package/merge.	Mistakes in libraries, management of change, or version control.
	Documentation.	Publication and maintenance notes.
	Algorithm.	Efficiency or correctness problems, corrected without change in design.
New	*Isolation.*	*Failure to isolate services and data of tenants.*
	IaaS / PaaS.	*Failures that arise from IaaS or PaaS configuration.*

- The bug report is usually based on the feedback that users submit to the SaaS provider's technical support or maintenance team, either via an online form, through email, or during a phone call. The margin of error in providing such feedback is considered somewhat medium, and the categorization of the defect may be misleading. Thus classifying defects based on the bug report would involve more data cleansing and analysis beyond the scope of this research.
- The application log contains too generic information that is closely related to the defects of the application which affect the operating system, thus it is not beneficial to our approach.
- The security log is mainly concerned with security

issues such as user log in or log out, or the attempts that have been made to breach the security of the application.

- The user session information is not applicable in our approach for two reasons. First, the session logging must be setup prior to using it, and developers rarely request it from the web server administrators. Second, the session contains only two generic fields, the first will store the user's information, and the second will store any data the developer wants to save in the post backs between the client and the server.
- The web server stores information about requests made to each website and its result in a log file. This information could be used to determine any malfunction in the website, such as bad formed requests, or missing files. The log file is composed of entries that contain information related to that request. Figure 3 depicts a sample entry in a web server log file. For example the request date, the status of the request, the referring page (see `http://technet.microsoft.com/en-us/library/cc786596(v=WS.10).aspx`). Table 3 explains the fields that are included in each entry.

Based on our findings, the most suitable resource is the web server log. Adopting the same methodology implemented in [10] with some modification will allow us to discover and classify SaaS defects efficiently. These other data sources will be explored in followup research to further characterize the discovered defects with additional details.

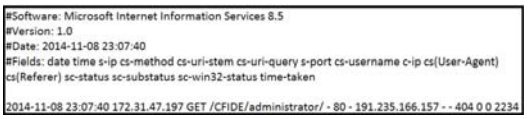


Fig. 3: Sample entry in a web server log file.

3.3 Data Processing for Cloud-ODC

One of the most important fields to the proposed approach is the "Protocol Status" field. Information stored in this field allows us to distinguish between successful and unsuccessful requests, which will lead us to defects. Protocol Status codes range from 100 to 599 (see `http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml`). Table 4 explains these codes. Status codes that are within the 4xx and 5xx groups are most important to the proposed new method. The 4xx status codes refer to defects that are caused by the client which is the SaaS layer in the proposed approach. And the 5xx status codes refer to defects that are caused by the server, which is the IaaS/PaaS layer in the proposed approach.

Table 3: Fields in a web server log file entry.

Field	Description
Date.	The date on which the activity occurred.
Time.	The time, in coordinated universal time (UTC), at which the activity occurred.
Client Ip Address.	The IP address of the client that made the request.
User Name.	The name of the authenticated user who accessed your server. Anonymous users are indicated by a hyphen.
Service Name and Instance Number.	The Internet service name and instance number that was running on the client.
Server Name.	The name of the server on which the log file entry was generated.
Server IP Address.	The IP address of the server on which the log file entry was generated.
Server Port.	The server port number that is configured for the service.
Method.	The requested action, for example, a GET method.
URI Stem.	The target of the action, for example, Default.htm.
URI Query.	The query, if any, that the client was trying to perform. A URI query is necessary only for dynamic pages.
Protocol Status.	The HTTP status code.
Protocol Sub status.	The sub status error code.
Win32 Status.	The Windows status code.
Bytes Sent.	The number of bytes that the server sent.
Bytes Received.	The number of bytes that the server received.
Time Taken.	The length of time that the action took, in milliseconds.
Protocol Version.	The protocol version - HTTP or FTP - that the client used.
Host.	The host header name, if any.
User Agent.	The browser type that the client used.
Cookie.	The content of the cookie sent or received, if any.
Referrer.	The site that the user last visited. This site provided a link to the current site.

3.4 Cloud-ODC Classification

Analyzing web server log file fields provides us with valuable information about the observed defects that will allow us to classify them. Below is an explanation of how the information in these fields aid in classifying the defects to the Cloud-ODC attributes:

- Information in the "date" and "time" fields can be used to group defects.
- Information in the "Protocol status" field allows classifying defects based on the defect type attribute.
- Information in the "Referrer" field allows classifying defects based on the trigger attribute.
- Extracting the file name from the "URI stem" field allows classifying defects based on the source attribute.
- Combining information from the "URI stem" field, the "User name" field (if known and available), and the service rented to the tenant allows knowing what service was effected, and based on that it is possible

Table 4: Protocol status codes explanation.

Status code	Purpose
1xx	Informational.
2xx	Success.
3xx	Redirection.
4xx	Client error.
5xx	Server error.

to classify defects based on the Impact and Severity attributes.

There are two more fields that provide us with more insight and information in the classification process, the "Protocol Sub status" field, and the "Win32 Status" field. Information from both allows us to better classify the type of the defect. For example, if a defect had a protocol status code of "404", and a Win32 Status of "2", the defect type would probably be listed as a "Function" defect. A similar defect with the same protocol status code "404" and a Win32 Status of "64" would probably be listed as an "Interface" defect (see `http://support.microsoft.com/kb/943891/en-us`, and `http://www.drowningintechnicaldebt.com/RoyAshbrook/archive/2007/10/04/http-status-and-substatus-codes.aspx`).

Table 5: Severity classification attributes.

Severity
Critical.
High.
Medium.
Low.

The severity of defects will be classified as critical, high, medium, and low. As shown in Table 5. As for classifying defects for the impact attribute, there were two standards to choose between. Either choose the standards developed by ISO (International Organization for Standardization) which are related to software engineering and quality (ISO 9126, and ISO 25010). Or choose IBM's measurement areas for quality attributes CUPRIMD (Capability, Usability, Performance, Reliability, Installibility, Maintainability, and Documentation) [11]. For the proposed approach, IBM's standards was chosen to allow for comparison with the original ODC work. Moreover, since security is an essential part in any web application, a new attribute called "Security" was added to the standard. As for the metric area of the new attribute, the sub categories of the security category in the ISO 25010 standard were used. Thus enabling the standard to be used for impact classification. Table 6 shows the impact classification attributes. The newly introduced security attribute is in *italic*.

Table 6: Impact classification attributes.

Attribute	Metric Areas
Capability.	Functionality delivered versus requirements.
	Volume of function to deliver.
Usability.	Ease of learning important tasks.
	Ease of completing a task .
	Intuitiveness.
Performance.	Transaction throughput.
	Response time to enquiry.
	Size of machine needed to run the product.
Reliability.	Mean time between failures.
	Number of defects.
	Severity of defects.
	Severity/impact of failures.
Installibility.	Ease of making product available for use.
	Time to complete installation.
	Skill level needed by installer.
Maintainability.	Ease of problem diagnosis.
	Ease of fixing problem correctly.
Documentation.	Ease of understanding.
	Ease of finding relevant information.
	Completeness of information.
Security.	*Confidentiality.*
	Integrity.
	Non-repudiation.
	Accountability.
	Authenticity.
	Compliance.

4. Case Study

To verify the viability of the new approach, a SaaS was installed on an Amazon EC2 (Elastic Computing Cloud). The SaaS is distributed as an open source CMS (Content Management System). From within the SaaS we created several portals, each corresponding to a tenant. Each portal had several users. After running the SaaS for one week, we collected the web server log files and examined it for defects. The log files contained 77475 entries, 342 of which contained a 4xx and 5xx error code. These error codes as described above are an indication of defects.

Table 7: Classifying for the defect type attribute.

Error code	Win32 status code	Defect type
404	0	Interface.
404	2	Function.
404	64	Interface.
404	1236	Interface.
500	All	IaaS/PaaS.

In order to benefit from the ODC concept, the observed defects need to be classified based on the frameworks attributes. For example, to classify defects based on the defect type attribute we need to know the error code and the Win32 status code. Having a specific combination of these codes results in a specific classification. If a defect has an error code of 404 and a Win32 status code of 2, the defect type is then classified as a function defect. Table 7 shows an example of classifying defects for the defect type attribute.

Table 8: Classifying for the layer affected attribute.

Error code	Layer affected
404	SaaS.
500	IaaS/PaaS.

Another example is classifying defects based on the layer affected attribute. If the error code was within the 4XX range then the defect is classified as a SaaS layer defect. On the other hand, if the error code was within the 5XX range, then the defect is classified as an IaaS/PaaS layer defect. Table 8 shows an example of classifying defects for the layer affected attribute.

5. Analysis of Results and Discussion

The examples in Figures 4, 5, 6, 7 are a demonstration of one-way classification of the observed defects in the web server log file based on the attributes: defect type, impact, layer affected, and severity. One-way classification is the process of analyzing observed defects based on one classification attribute.

As stated previously, multi-tenancy and isolation are the major characteristics that distinguish between SaaS and traditional software. Since the original ODC does not cater to both of these characteristics, it is difficult to take advantage of the concept to provide in-process defect feedback to the development team. In our work we modified the existing ODC framework to better accommodate the new SaaS characteristics.

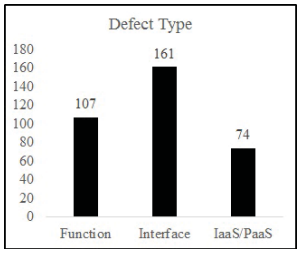


Fig. 4: One-way classification of the "Defect type" attribute.

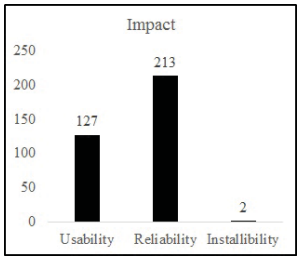


Fig. 5: One-way classification of the "Impact" attribute.

Figure 4 is a representation of classifying defects based on the defect type attribute. 107 of the observed defects were classified as functional defects, 161 were classified as interface defects, and 74 were classified as IaaS/PaaS defects which is one of the newly introduced defect types. In addition, Figure 5 is a representation of classifying defects based on the impact attribute. 127 of the observed defects were classified as defects related to usability, 213 defects as related to reliability, and 2 defects as defects related to installibility. Moreover, Figure 6 is a representation of classifying defects based on the layer affected attribute. It is clear that 268 defects affected the SaaS layer and 74 defects affected the IaaS/PaaS layer. Finally, Figure 7 is a representation of classifying defects based on the severity attribute. from the collected data the severity of 17 defects were classified as critical, 102 were classified as high, 98 were classified as medium, and 125 were classified as low severity.

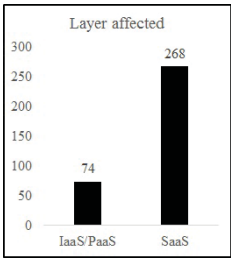


Fig. 6: One-way classification of the "Layer affected" attribute.

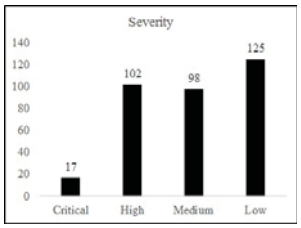


Fig. 7: One-way classification of the "Severity" attribute.

One-way classification may be informative but may not provide sufficient information regarding defects and related

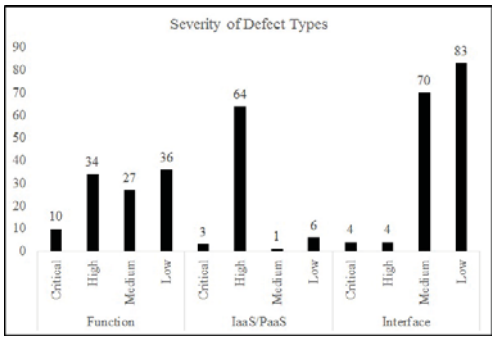


Fig. 8: Two-way classification of the "Defect type" and "Severity" attribute.

high-risk areas to the development team. Implementing two-way classification will result in gaining a better view of the observed defects, and eventually may result in better informed decisions. For example, Figure 8 shows the number of defects observed after implementing the two-way classification method based on the defect type and severity. It is clear that 10 of the observed defects are classified as Functional defects (based on the defect type attribute) and are also classified as critical (based on the severity attribute). Therefore, using two-way classification it is possible to focus on resolving high priority defects to improve the reliability and quality of the SaaS.

In our work, we have demonstrated that adopting the ODC model to a SaaS is feasible. The new approach aids in classifying and analyzing defects that may have not been classified or analyzed using regular testing techniques. Thus enabling SaaS providers to classify and analyze defects while considering the special characteristics of SaaS such as isolation and multi-tenancy. This will be an opportunity for SaaS providers to improve their product, and eventually lead to improved SaaS quality.

In addition, we also demonstrated the possibility of implementing one-way and two-way classification of observed defects. Such classification allows SaaS providers to focus on the most important defects that significantly affect the reliability of the SaaS.

6. Conclusions

Locating, analyzing, and resolving defects is considered a major contributor to improving software quality. ODC is a defect analysis method, known for providing In-process defect feedback to the development team. However, there has not been a clear defect analysis model designed specifically for SaaS. A SaaS that contains defects is considered with low quality, and eventually may not comply with the level of quality promised in the SLA. There has been several ways and models developed to analyze defects. However, none is applicable to SaaS due to the fact that these techniques and models do not address the multi-tenancy and isolation features of SaaS.

The defect analysis method described in this paper is based on adopting the ODC concept to a SaaS running in the Cloud. The dynamic structure of the Cloud requires a dynamic defect analysis approach that allows benefiting from the availability of feedback loops in most software development life cycles. Adopting the Cloud-ODC framework allows notifying the development team of discovered defects through these feedback loops, which is considered a major contributor to enhancing the overall quality of the software product. Therefore it is beneficial to SaaS providers.

Using the one-way or two-way analysis of the classified defect data enables focusing on certain defects to improve the quality of the SaaS by fixing or resolving the most important defects. However, there are limitations to our approach. For example, the experience of the individual that is conducting the classification plays a role in the classification results. In addition, this approach has not been tested towards a realistic industry-strength Cloud application, which we plan to do in follow-up studies. Moreover, since the case study was conducted using the web server developed by Windows, the “Win32 status” field is not applicable to other web servers. However, other web servers have similar fields that can be utilized.

In Our work, we have demonstrated the viability of adopting the ODC concept to characterize SaaS defects. We modified the current ODC framework to better accommodate the multi-tenancy and isolation features of a SaaS, and added the “Layer affected" attribute to the proposed framework. In addition, we added the “security” attribute to IBM’s measurement areas for quality attributes. We also implemented one-way and two-way classifications successfully on a running SaaS in the Cloud. This would enable SaaS providers to focus on certain defects with high priority, which would promise cost effective quality improvement to the service they provide to their customers.

Acknowledgment

This research is supported in part by NSF Grant #1126747, NSF Net-Centric I/UCRC, and the Institute of Public Administration (IPA) in the Kingdom of Saudi Arabia.

References

[1] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal defect classification - a concept for in-process measurment,” in *IEEE Transactions on Software Engineering*, vol. 18, no. 11, November 1992.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” in *Communications of the ACM*, vol. 53, no. 4, April 2010, pp. 50–58.

[3] The National Institute of Standards and Technology, “The NIST definition of cloud computing, Special Publication 800-145,” 2011.

[4] A. Ojala, “Software renting in the era of cloud computing,” in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 662–669.

[5] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, “A framework for native multitenancy application development and management,” in *The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, ECommerce, and E-Services*, 2007.

[6] Y. Zhu and J. Zhang, “Research on key technology for saas,” in *The 7th International Conference on Computer Science & Education (ICCSE) 2012*, Melbourne, Austrailia, July 2012.

[7] J. Tian, *Software Quality Engineering*. John Wiley & Sons., 2005.

[8] B. Boehm, “Software risk management: principles and practices,” *Software, IEEE*, vol. 8, no. 1, pp. 32–41, Jan 1991.

[9] J. Liu, Z. Xu, J. zhong, and S. Lin, “A defect prediction model for software based on service oriented architecture using expert cocomo,” in *2009 Chines Control and decision Conference (CCDC 2009)*, 2009, dOI 1978-1-4244-2723-9/09.

[10] L. Ma and J. Tian, “Web error classification and analysis for reliability improvement,” in *The Journal of Systems and Software*, vol. 80, 2007, pp. 795–804.

[11] R. A. Radice and R. W. Phillips, *Software Engineering: An Industrial Approach*. Prentice-Hall, 1988.

Support for Security Analysis of Design Models based on Traceability

Hirokazu Yatsu[1], Masaru Matsunami[2], Toshimi Sawada[3], Go Hirakawa[4], Atsushi Noda[4], Naoya Obata[4], Takahiro Ando[1], Kenji Hisazumi[1], Weiqiang Kong[5], and Akira Fukuda[1]
[1]Kyushu University, Fukuoka, Japan
[2]Sony Digital Network Applications, Inc., Tokyo, Japan
[3]Kosakusha, Ltd., Tokyo, Japan
[4]Network Application Engineering Laboratories, Ltd., Fukuoka, Japan
[5]Dalian University of Technology, Dalian, China

Abstract - *Software systems embedded into the foundation of information society is required to be secure. Requirements for the system to be secure should be properly recognized in the upper process of system development, and accurately reflected in their specifications and designs. However, security analysis to decide whether systems are secure or not is usually done at the implementation phase of system development or later. In this paper, we propose a universal approach to support security analysis at the design phase. Our approach is to detect vulnerable parts of systems based on traceability established among SysML diagrams, security threats and countermeasures against threats using SMT solvers.*

Keywords: Security Analysis, Traceability, SysML, FTA, GSN, SMT solver

1 Introduction

Software systems embedded into the foundation of information society is required to be secure. Requirements for the systems to be secure (in this paper, this is abbreviated to security requirements) should be properly recognized in the upper process of system development, and accurately reflected in their specifications and designs. However, in the current system development, security analysis to decide whether systems are secure or not is mainly performed on code created at the system implementation phase with tools, such as vulnerability scanning on binary code or static analysis on source code. It might be necessary to correct the specifications and design of the systems to remove the security vulnerabilities found through security analysis at the implementation phase or later. In that case, the overhead to correct the specifications and design would be large. Therefore, security analysis on the system specifications and design (in this paper, this is abbreviated to security design analysis) is necessary. In security design analysis, we have to grasp how countermeasures against security threats are prepared in the systems. Unfortunately, the necessity and importance of security design analysis have not been recognized yet.

In this paper, we introduce the attempt of security design analysis in Sony Digital Network Applications, Inc. (hereinafter abbreviated to SDNA Inc.) [1] and our tool to support the security design analysis by detecting vulnerable parts of the systems using SMT solvers.

2 Security design analysis

In this section, as an example of the security design analysis, we will introduce the attempt of SDNA Inc. [1]. This attempt refers to the threat modeling [2]. The subject of threat modeling is security threats with which a system is facing, whereas the subject in [1] is situations where assets of a system such as confidential materials, password, etc., are protected from security threats.

2.1 Threat modeling

The threat modeling draws attack trees, whose structure is same as fault trees drawn in Fault Tree Analysis (FTA for short) (Figure 1). However, representation of the security threats in attack trees varies from person to person. It would be difficult even for practitioners rich in security knowledge to draw attack trees.

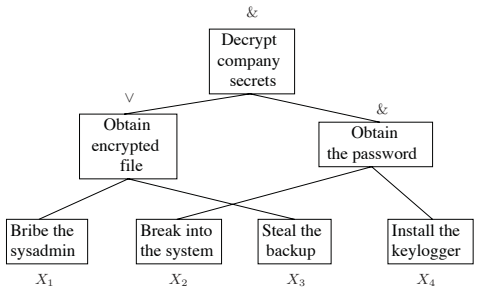


Figure 1 An example of attack tree[3]

2.2 Security design analysis

As security requirements, [1] describes a desirable situation where assets of a system are protected from security

threats. The assets include information and functions stored in the system. An example of the situation is that a third person cannot read/call such information/functions through any security attack. In [1], security threats that cannot be overlooked in a system are identified as the following two actions of a third person.

1. Access to assets
2. Prevention of user's access to assets

Security requirements of a system assume that assets of the system are always protected from any security attacks, that is, any third person cannot access assets of the system and cannot prevent user's access to the assets. In [1], a security threat is not directly described, but a security requirement is described as a dual(i.e., negation) of existence of some security threat.

Same as the case of drawing attack trees, representation of security requirements are likely to vary from person to person. So, [1] provides a template (Table 1) for description of security requirements. This template provides practitioners a uniform description style for security requirements. So it enables them to represent security requirements similarly to other practitioners.

Table 1 A template for description of security requirements

Asset	Action	Security requirement
Information	read	<Attacker> cannot read <Information>
		<Attacker> cannot prevent <Users> from reading <Information>
	write	<Attacker> cannot write <Information>
		<Attacker> cannot write <Information>
Function	execute	<Attacker> cannot write <Information>
		<Attacker> cannot prevent <Users> from executing <Function>

2.3 Decomposition of security requirements

Security requirements can be decomposed according to the configuration of a system. Figure 2 shows an example of a system which manages prescriptions assigned to patients. This system is called "Medication Notebooks System"[1]. In the system, all prescription records registered through receipt computers in pharmacies are gathered in a server. Dotted lines indicate the flow of prescription records in the system. Numbered constituents such as server, receipt computer, tablet, etc. indicate locations where prescription records exist. One of the security requirements of this system is "prescription records cannot be read by a third person in the system".

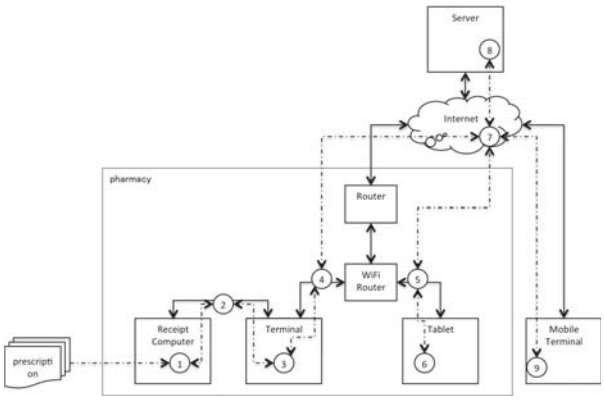


Figure 2 Flow of a prescription

That security requirement is equivalent to "prescription records cannot be read by a third person at any location of the system where they exist", which is represented as a conjunction indicated in the Figure 3.

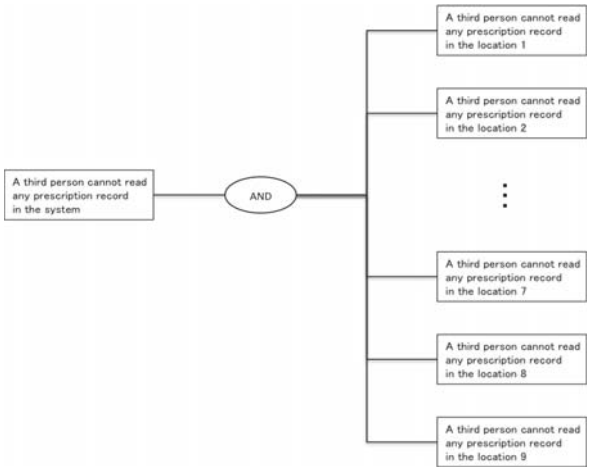


Figure 3 An example of decomposition

3 Support of security design analysis base on traceability

A security requirement for a system that the system is secure can only be guaranteed by indicating that assets in the system can be protected from security threats already found. As described in the previous section, security requirements derived from the security design analysis can be decomposed in accordance with the configuration of the system. As a security requirement is a dual of some security threat, the threat can also be decomposed in accordance with the configuration of the system. So security requirements and security threats should be recognized in association with the configuration of a system that is a target of security design analysis. Appropriate traceability should be established among them.

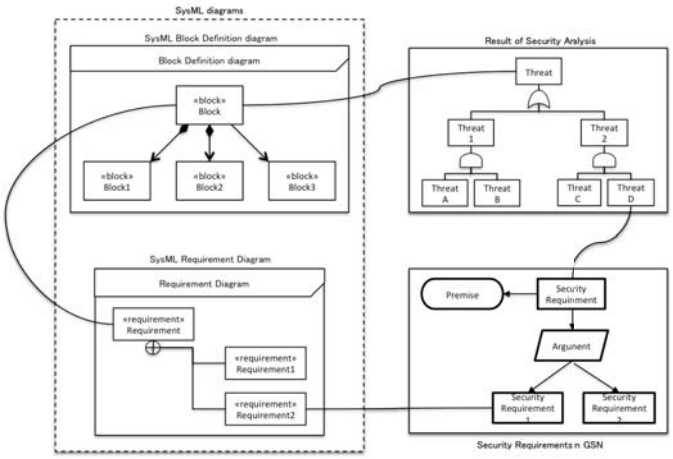


Figure 4 An example of established traceability

Figure 4 shows an example of traceability established among configuration of a system, security threats and security requirements. Configuration of a system is represented by SysML[4] block definition diagrams and requirement diagrams. Security threats are described as constituents which appear in a fault tree(or an attack tree). Security requirements are described in Goal Structuring Notation(GSN for short)[5]. The established traceability is interpreted as in Table 2.

Table 2 The interpretation of traceability

Traceability between		Interpretation
block B	threat T	block B is faced with threat T
threat T	security requirement M	If security requirement M is satisfied, then threat T does not occur
security requirement M	requirement (in SysML) R	If requirement R is satisfied, security requirement M is also satisfied
requirement R	block B	block B satisfies requirement R

3.1 Detection of vulnerable parts in a system

We define that a vulnerable part of a system is a block which does not satisfy enough security requirements to prevent security threats from occurring on the block. In this section, we describe a mechanism to detect vulnerable parts in a system.

Suppose that traceability is established among a block B, a threat T, a security requirement M and a requirement R(see Figure 5).

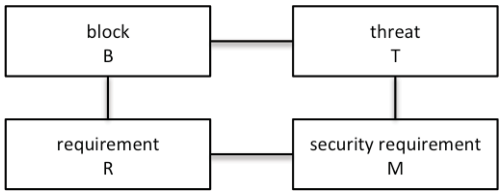


Figure 5 An example of detection

Then, some propositions can be derived using the interpretation of traceability in Table 2, same propositions can be extracted (Table 3).

Table 3 Extraction of propositions

Interpretation	extracted proposition
block B is faced with threat T	B implies T
If security requirement M is satisfied, then threat T does not occur	M implies not T
If requirement R is satisfied, security requirement M is also satisfied	R implies T
block B satisfies requirement R	B implies R

B, T, M and R in the extracted propositions are propositional variables which denote 'existence of block B', 'occurrence of threat T', ' satisfaction of security requirement M' and 'satisfaction of requirement R', respectively.

If block B is not vulnerable, there is no assignment to the propositional variables B, T, M and R. From the last three propositions in Table 3, we can deduce that B implies not T, which contradicts the first proposition B implies T. If block B is vulnerable, there is some assignment F to B, T, M and R such that F(T) = true. We can easily decide whether there is such assignment or not using SMT solvers, for example Z3[6] and Yices[7].

3.2 An assistant to detect vulnerable parts

In this section, we introduce our tool which assists detection of vulnerable parts in a system. This tool extracts propositions from SysML diagrams(block definition diagram and requirement diagram), fault/attack trees of security threats and GSN expression of security requirements. Propositions extracted by the tool are written in SMT-LIBv2[8] which is a standard input language for SMT solvers.

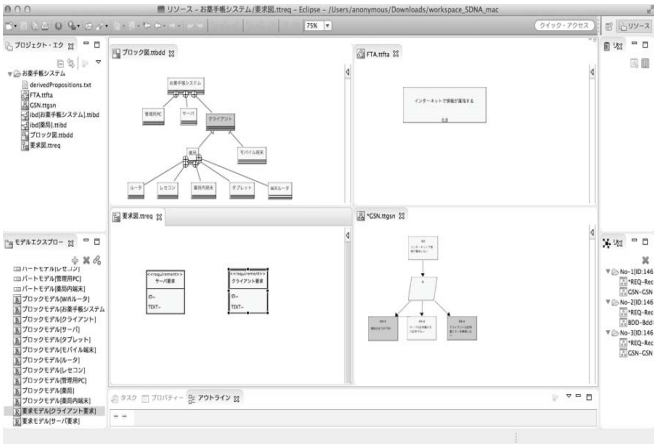


Figure 6 Our assistant tool for detection of vulnerable parts

The above Figure 6 is a screen of our tool, where SysML diagrams, a fault tree of a security threat and a GSN expression of a security requirement are shown. From these information appear in the screen, our tool extract propositions, which is precisely described in the following section.

3.2.1 Inheritance relation between blocks

Suppose that block B_2 inherits block B_1 (Figure 7).

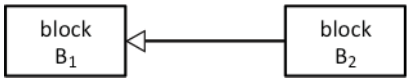


Figure 7 Inheritance relation between blocks

Then the following proposition (1) is extracted.

$$B_2 \Rightarrow B_1 \tag{1}$$

3.2.2 Traceability between a block and a threat

Suppose that traceability is established between a block B and a threat T (Figure 8).

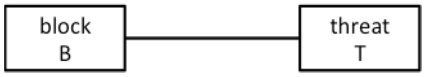


Figure 8 Traceability between a block and a threat

Then the following proposition (2) is extracted.

$$B \Rightarrow T \tag{2}$$

3.2.3 Causation relation between security threats

Suppose that threat T is caused from threats T_1 and … and T_n (Figure 9).

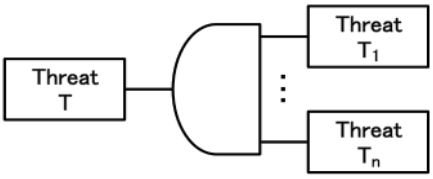


Figure 9 Causation relation between threats(AND)

Then the following proposition (3) is extracted.

$$T \Rightarrow (T_1 \wedge \ldots \wedge T_n) \tag{3}$$

Suppose that threat T is caused from threat T_1 or … or T_n (Figure 10).

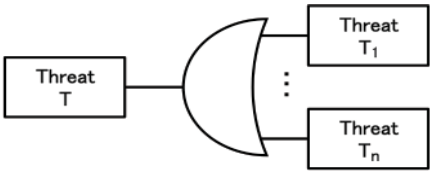


Figure 10 Causation relation between threats(OR)

Then the following proposition (4) is extracted.

$$T \Rightarrow (T_1 \vee \ldots \vee T_n) \tag{4}$$

3.2.4 Traceability between a threat and a security requirement

Suppose that traceability is established between threat T and security requirement M (Figure 11).

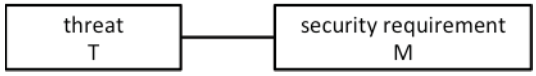


Figure 11 Traceability between a threat and a security requirement

Then, the following proposition (5) is extracted.

$$M \Rightarrow \neg T \tag{5}$$

3.2.5 Constitution of a security requirement

Suppose that security requirement M is decomposed to sub requirements M_1, … ,M_n under the premise P (Figure 12).

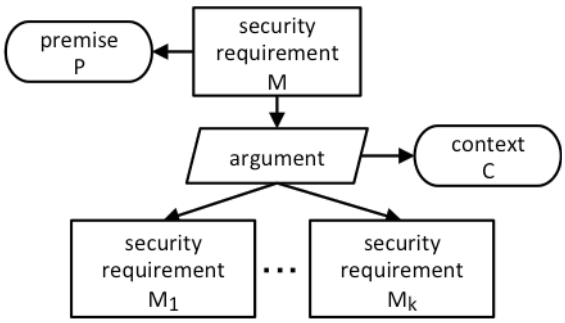


Figure 12 Constitution of a security requirement

Then, the following proposition (6) is extracted.

$$(P \wedge M_1 \wedge ... \wedge M_k) \Rightarrow M \quad (6)$$

3.2.6 Traceability between a security requirement and a requirement

Suppose that traceability is established between security requirement M and requirement R (Figure 13).

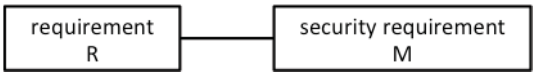


Figure 13 Traceability between a security requirement and a requirement

Then the following proposition (7) is extracted.

$$R \Rightarrow M \quad (7)$$

3.2.7 Relation between requirements

Suppose that requirement R is derived from requirement R_1 (Figure 14).

Figure 14 Derivation relation between requirements

Then the following proposition (8) is extracted.

$$R_1 \Rightarrow R \quad (8)$$

And suppose that requirement R is composed of R_1, …, R_n (Figure 15).

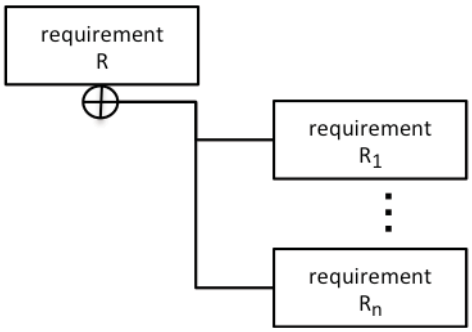


Figure 15 Composition of a requirement

Then the following proposition (9) is extracted.

$$R \Rightarrow (R_1 \wedge ... \wedge R_n) \quad (9)$$

3.2.8 Traceability between a block and a requirement

Suppose that traceability is established between block B and requirement R (Figure 16).

Figure 16 Satisfaction relation between a block and a requirement

Then the following proposition (10) is extracted.

$$B \Rightarrow R \quad (10)$$

4 Example

The validity of the proposed mechanism is checked by several examples. Figure 17 indicates one of such examples. This example treats the medication notebook system[1]. Pharmacies and mobile terminals in the system are instances of the block Client. In the example, a security threat is that prescription records can be read during a communication on the internet by a third person. A security requirement against the threat consists of three sub-requirements. In the following figure, dotted lines indicate traceability established in the system.

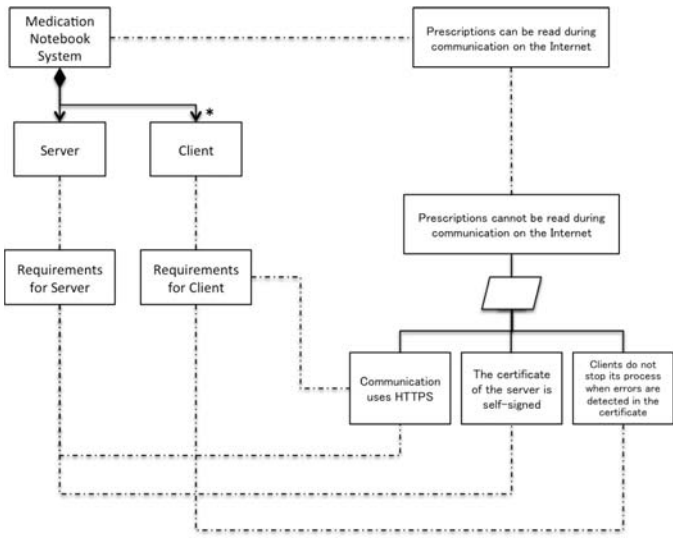


Figure 17 An example for checking validity of the proposed mechanism

According to the proposed mechanism introduced in the section 3, there is no vulnerable part in the example because each block satisfies enough security requirements to prevent the security threat. However, when some traceability among the threat, the security requirements and the requirements (in SysML requirement diagram) is removed, we can realize that the threat can occur on the system, because SMT solvers find a assignment where a propositional variable which represents the occurrence of the threat to be true.

5 Conclusions

We proposed a mechanism and introduced a tool to support security design analysis by detecting vulnerable parts of a system using SMT solvers. We are going to apply the mechanism to some systems of the real world to validate the usefulness of the proposed mechanism for security design analysis.

6 Acknowledgement

This research was supported by the Ministry of Internal Affairs and Communications SCOPE(Strategic Information and Communications R&D Promotion Programme) No. 142310011.

7 References

[1] Masaru Matsunami. "Security Design Analysis Method applied to Sony's Electronic Medication Notebooks System". proc. of WOCS 2015, 2015 (in Japanese).

[2] Frank Swiderski and Window Snyder. "Threat Modeling (Microsoft Professional)". Microsoft Press, 2004.

[3] Aivo Jurgenson and Jan Willemson. "Serial Model for Attack Tree Computations". proc. of Information, Security and Cryptology – ICISC 2009, Lecture Notes in Computer Science Volume 5984, 2010, pp 118-128.

[4] "OMG Systems Modeling Language Version 1.3", available at http://www.omg.org/ spec/SysML/1.3/PDF.

[5] "GSN COMMUNITY STANDARD", available at http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf

[6] http://research.microsoft.com/en-us/um/redmond/projects/z3/

[7] http://yices.csl.sri.com/

[8] David Cok. "The SMT-LIBv2 Language and Tools: A Tutorial", available at http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf

Automatization of Acceptance Test for Metrology Algorithms

B. Müller
Ostfalia – University of Applied Sciences
Faculty of Computer Science
Germany – 38302 Wolfenbüttel
bernd.mueller@ostfalia.de

Abstract— *Coordinate Measuring Machines use complex metrology algorithms to compute geometric shapes based on 3-dimensional measuring points. National metrology institutes are committed to validate that results computed by these algorithms are correct. We report on a project funded by the European Union which, beside other topics, develops criteria to assess the fitness for purpose of computational algorithms and software in metrology used by coordinate measuring machines.*

Keywords: Testing, verification, validation, TraCIM, coordinate measuring machines

1. Introduction

Coordinate Measuring Machines (CMM) are used in different manufacturing industries to ensure high accuracy of manufactured products but also high accuracy of the production run itself. The research project TraCIM (*Traceability for Computational-Intensive Metrology*), funded by the European Union, aims for the development of a coherent framework to ensuring traceability in computationally-intensive metrology, a basis for ensuring the trustworthiness and fitness for purpose of metrology software for coming decades. To reach this aim the software and therefor its underlying algorithms have to be verified and validated.

We first introduce CMMs (Coordinate Measuring Machines) and report on the current manual process to check for the correct and high precision measuring processes of such machines. Further, we classify the checking process with respect to the established software engineering concepts of verification and validation. Finally, we describe the architecture and functionality of a system which automates the whole process.

2. Coordinate Measuring Machines

CMM are devices for measuring physical geometrical characteristics of different kind of objects. Maximal permissable error is typically around $1\ \mu\mathrm{m}$. The high accuracy measuring can be achieved by optical, tactile or even computer tomography scanner based capturing of probes. CMMs are hardened against floor induced vibration and are operated in an air conditioned environment to prevent measuring errors.

The capturing of probes differ from conventional measuring. Substitution points get captured and represented as $x/y/z$ coordinates. Based on these substitution points the geometrical forms are computed. Figure 1 shows a circle and the captured substitution points in a plane.

In practice, 3-dimensional geometric bodies such as cubes or cylinders have to be measured and their surfaces or volumes have to be computed. In modern manufacturing industry high accuracy measuring is important to verify that manufactured parts are within designer-specified tolerances and to ensure that manufacturing processes stay in control during the production run.

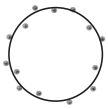


Fig. 1
SUBSTITUTION POINTS OF A CIRCLE

CMM manufacturers therefore have to implement algorithms in some programming language to compute, for example, the circle (diameter, circumference or circular area) depicted in figure 1 out of the substitution points. This can be done with different algorithms — for example least-square, Gaussian and Chebyshev algorithms — and, of course, different programming languages. For an introduction of CMM and used algorithms see [1], [2], [3].

3. Manual Certification Process

National Metrology Institutes (NMI) provide scientific and technical services for industry, science and society. For example, NMIs have to do some certification and support calibration of CMMs to support manufacturing processes of high technology industries.

At the moment the process of certification is done manually by NMIs around the world in a variety of ways. For example, some NMIs own test data sets, which represent substitution points as introduced in section 2. The test data is sent per

e-mail or ground mail (CD/DVD) to the requesting CMM manufacturer. The manufacturer uses the data as input for his metrology algorithms and sents the computed result back to the NMI, also per e-mail or ground mail. The NMI compares the computed result with the expected result and hands over a certificate if the computed and expected results match within some tolerances.

This manual certification process is lenghty, error prone and expensive because of the great portion of human work. Process automation is therefor an evident demand. However, there exist many more requirements, for example concerning traceability which we will detail on in section 5.4.

4. Verification and Validation in Software Engineering

While there is some confusion regarding the terms and definitions of verification and validation across different siences Boehm defines already 1979 in his seminal paper [4] the terms with respect to software engineering:

Verification: to establish the *truth* of the correspondence between a software product and its specification. ('Am I building the product right?')

Validation: to establish the *fitness* or *worth* of a software product for its operational mission. ('Am I building the right product?')

In the figure called *V-Chart* following the above definitions Boehm depicts that verification is done regarding formal requirements while validation ist done regarding customer expectation.

In a complete product development life cycle the transition from validation to verification and vice versa is fluent by nature. Relating to CMM the customer expectation of correct and exact measurement, the implementation of Gaussian and Chebyshev algorithms from the 19th century with some programming language and the embedding into some physical machine has to function correctly as a whole. At the very end there is some last acceptance test resulting in adoption or refusal of the product.

From a software engineering point of view component and integration tests are fully automated while system and acceptance tests are not. It is therefore helpful to look for the characteristics of component and integration tests:

- Test method knows the method to test.
- Test method calls method to test. Both are written in the same programming language.
- Test result is undoubtful.
- Test motivation, test coverage etc. are defined by project conditions.

In contrast TraCIM tests are characterized by:

- Method to test respectivly the environment knows the test data set or test data generator.
- Test is executed randomly and application specific.
- Method or algorithm to test written in some programming language has to obtain test data self-dependent.
- Test result is supposed to be correct but this is not ensured.
- Successful tests lead subseqently to some certification. Therefore test motivation, test coverage etc. are defined by public authorities.

5. Process Automation

Despite long history in formal verification research [5] only very small and simple software systems can be verified correct based on formal methods of mathematics. In practice the only valid choice to get some confidence in proper software operation is testing as introduced in section 4. Some NMIs own test data sets with corresponding test solutions. Some NMIs generate test data sets on the fly and the test solutions are computed, too.

The TraCIM Software Verification System (TraCIM SVS) is part of the TraCIM project (Traceability for computational-intensive metrology). We depict the project further in section 6. A detailed description is also available online [6].

From a software engineering point of view the requirements for TraCIM SVS are quite standard:

- Clients, humans or other software systems ask for some (test) data
- After the data is received some computation regarding the computational coordinate metrology algorithms from section 2 takes place
- The resulting new data (the test result) is send back to the system as the solution for the test data
- After verification of the submitted data there is some kind of result, either success or failure

One of the most popular environments to implement such systems is the Java Platform Enterprise Edition (Java EE) [7]. TraCIM SVS is build with Java EE 7, the most current version. Java EE includes different parts, for example JavaServer Faces (JSF) to build HTML and Web based UIs, Enterprise JavaBeans (EJB) to implement business logic, Java Persistence API (JPA) to persist data to relational databases, JAX-RS to offer REST-like APIs and Context and Dependency Injection (CDI) to glue all the parts together.

The most important technical requirement of TraCIM SVS is the ability to handle all kind of tests, not only the 3D coordinate measurements features described in section 2. Therefore, TraCIM SVS consists of a core system and an innovative extension mechanism illustrated in the next section.

5.1 Architecture and Base Functionality

Figure 2 represents the main components of TraCIM SVS together with the client applications built by the CMM manufacturer. The TraCIM Server core offers REST based services and is hosted by a NMI. Functionality and communication steps are as follows

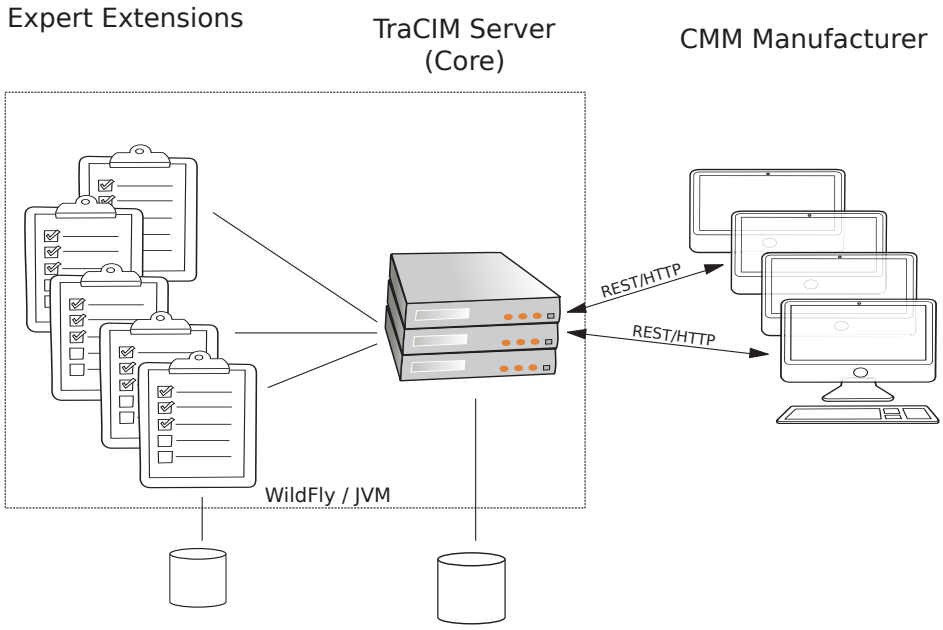


Fig. 2
TRACIM ARCHITECTURE

1) The client has payed the invoice and received some key, which enables him to request for test data. The key encodes also the type of test.
2) The client request TraCIM SVS for test data.
3) Because the request includes the key generated in step 1, TraCIM SVS is capable to identify the requested expert module. This special expert module is called and returns the test data.
4) TraCIM SVS sends back the test data as HTTP response.
5) The client computes the result for the received test data and sends the result back to the TraCIM SVS per HTTP request.
6) TraCIM SVS calls the expert module to compare the expected result for the provided test data and the actual result from the client. This comparison can succeed or fail. In both cases the result of the method call is returned to TraCIM SVS and includes a certificate in PDF in case of success.
7) TraCIM SVS returns the comparison result to the client. TraCIM envelopes the expert extension generated PDF with some administration information from the involved NMI.

As depicted in figure 2 the server stores management information in a database. The kind of management information ranges from CMM manufacturer identification, payment information and the number of remaining tests to memorandums which test data set was delivered to which client, including the time of test data delivery and the time of result submission of the client.

As mentioned earlier expert extension can generate test data on the fly but can also manage a set of static test data and expected test results stored in a database. This optional database usage is also depicted in figure 2. TraCIM SVS does not restrict in any case the inner working of expert extensions.

5.2 Implementation and Used Technologies

Java EE is a well known technology in the area of big application implementation and in widespread use. Java EE is an umbrella specification and consists of about 30 single specifications, depending on the version used. Our project started with version 6 of Java EE but was migrated to Java EE 7 in course of the project. Java EE implementations manifest themselves in so called *application servers*. There are many companies which offer application servers, for example

WebLogic® from Oracle, WebSphere® from IBM and JBoss-AS/WildFly from JBoss. We use JBoss-AS — which was renamed to WildFly in the last version — because JBoss-AS is a so called *open source* implementation of Java EE. Therefore, there are no costs of purchase as well as no annual subscription costs. If in later project stages some demand for commercial assistance will arise, Red Hat the parent company of JBoss offers a commercial licence called EAP which can be subscribed to.

TraCIM server core and expert extensions are implemented as Java EE applications. Because the server core has a JSF based UI it is deployed as a WAR (Web Archive). The expert extensions are deployed as JARs (Java Archive). Both, server as well as expert extensions use some common set of classes. To prevent code redundancy a so called extension base is the third kind of Java EE application we use and is also deployed as a JAR.

The Java EE standard dictates absolute separation of different applications to prevent negative impact from one application to another in case of malfunction. In our case we have the demand that some application modules use some other modules which is not an uncommon requirement in big applications. All application servers offer some kind of non standard mechanism to allow modules to access modules from different applications. This mechanism is usually based on Java's classloader architecture. In TraCIM SVS classes from the extension base are used by the server as well as by the expert extensions. The expert extensions are additionally used by the server.

Finally, TraCIM SVS consists of

- the extension base
- the server core
- one or more expert extensions

If a CMM manufacturer wants his software and in turn the complete CMM to get certified, he has to pay the mandatory fee for responsibilities of public administration and get the authorization to get test data sets and submit in turn test results for these data sets.

This is done by REST requests (REpresentational State Transfer), the most up-to-date interpretation of web services. The details about the communication steps are already described in section 5.1.

The most innovative aspect of the system architecture is based on the extension mechnism for expert extensions which is similar to plug-in architectures. If a new expert extension is implemented and has to be integrated into the system, no code change has to be accomplished. This is possible because of Java's concept of a *service loader* which was introduced in Java 6 and manifests itself in the class `ServiceLoader` [8]. The mechanism is based on a simple convention which results in a self publication of classes implementing a particular interface. The class `ServiceLoader` can then be asked for all known implementation of the particular interface.

5.3 Future Enhancements

Because we describe here some work in progress there will be of course future enhancements. At the moment we are working on a design enhancement to allow expert extensions to run as separate server services. If, for example, some NMI X hosts the TraCIM server core but the expert extension runs on behalf of NMI Y on a different server, probably in a different country the collaboration of TraCIM server core and expert extensions has to be revised to reflect this requirement. The generated certificate has also to reflect this separation of responsibilities. It has to contain a functional part of the NMI offering the expert extension but also a more administration part of the NMI hosting the TraCIM server core which reflects the contractual relationship between the NMI and the CMM manufacturer. This point directly passes over to some legal aspects.

5.4 Legal Aspects

Because certificates were assigned by public authorities there are some legal consequences. The performed tests and certifications have to be repeatable and traceable. Repeatable means that if a CMM manufacturer has requested a particular kind of test and has succeeded this test a consumer of the CMM can ask many years later for a further test. It has to be guaranteed that the consumer will get the same test data set as the manufacturer many years before to ensure that the outcome of the same submitted test results are the same.

Based on the same rationals and the responsibilities of public administration all test processes and test results have to be stored for decades to establish a complete chain of evidence if some disaster happens because of some earlier certification of wrong or even right test results.

6. Project and Project partners

The European Community has established the research project *Traceability for Computationally-Intensive Metrology* (TraCIM) which - beside other topics - develops criteria to assess the fitness for purpose of computational software in metrology and to verify them. The TraCIM home page [6] details objectives of this research project. The project started in 2013 and will be finished in 2015.

The national metrology institutes of the United Kingdom, Czech Republic, Italy, Germany, Slovenia and Netherlands as well as 4 CMM manufactures and 3 Universities belong to the project consortium.

Ostfalia, University of Applied Sciences, located in Germany is responsible for implementing the project supporting software and therefore TraCIM SVS. Close collaboration takes place with PTB, the German NMI.

Some NMIs are working on different expert extensions at the time to complete the bunch of possible test data sets for different aspects of CMM characteristics. At the moment

Guassian, Chebyshev and Intercomparison are available and offered by PTB.

7. Conclusion

We reported on the software system TraCIM SVS which main task is to support national metrology institutes to proof and certify the correct working of CMM. CMM are an important part of manufacturing processes in modern industry.

Workshops in fall 2013 and spring and fall 2014 with the CMM manufacturers of the TraCIM project demonstrated the capacity of the design and implementation path we have chosen. At the moment two manufacturer's CMM software was certified by PTB, the German NMI, in a fully automatized process based on TraCIM SVS.

Acknowledgment

This research was undertaken within the EMRP project NEW06-REG3 TraCIM. The EMRP is jointly funded by the EMRP participating countries within EURAMET and the European Union.

References

[1] V. Srinivasan and C. M. Shakarji and E. P. Morse, *On the Enduring Appeal of Least-Squares Fitting in Computational Coordinate Metrology*, Journal of Computing and Information Science in Engineering, March 2012, Vol 12.

[2] T. H. Hopp and M. S. Levenson, *Performance-Measures for Geometric Fitting in the NIST Algorithm Testing and Evaluation Program for Coordinate Measurement Systems*, Journal of Research of the National Institute of Standards and Technology, 9/1995.

[3] C. Shakarji, *Evaluation of one- and two-sided Geometric Fitting Algorithms in Industrial Software*, Proc. American Society of Precision Engineering Annual Meeting, 2003.

[4] B. W. Boehm, *Guidelines for verifying and validation Software requirements and design specification*, Proc. European Conference of Applied Information Technology of the International Federation for Information Processing, London, 1979.

[5] E. M. Clarke and E. A. Emerson and A. P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems, Vol 8, Apr. 1986.

[6] *Traceability for Computationally-Intensive Metrology*, http://www.ptb.de/ emrp/1389.html.

[7] *Java Platform, Enterprise Edition*, http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[8] *ServiceLoader Documentation*, http://docs.oracle.com/javase/7/docs/api/java/util/ServiceLoader.html

Design and Implementation of a Constraint-Based Test Case Generator with Test Fixture Computation

Chi-Kuang Chang and Nai-Wei Lin
Department of Computer Science and Information Engineering,
National Chung Cheng University, Chaiyi, 168, Taiwan, R.O.C.

Abstract— *In this study, we design and implement a constraint-based test case generator. Test cases for method-level unit-testing can be generated automatically from UML class diagrams and OCL specifications. A test case includes test data (test inputs and expected outputs), and test fixtures. We adopt a constraint logic graph approach to generating test data, and a finite model reasoning approach to generating test fixtures. These two approaches are unified and resolved by using constraint logic programming. We have also performed a preliminary experiment to evaluate the applicability of this test case generator.*

Keywords: constraint-based testing; UML/OCL; method-level unit testing; constraint satisfaction problem; constraint logic programming.

1. Introduction

Manual acquisition of test cases is laborious, time-consuming, and error-prone. It is beneficial if test cases can be automatically generated instead. Constraint-based testing (CBT) is one of the approaches to generate test cases from formal specifications, against a testing objective, by means of constraint solving techniques [1]. In this study we propose a constraint-based approach to generate platform-independent test data from the software models. Test cases for different programming languages and test-platforms can then be generated from the test data.

Method-level unit-testing focuses on verifying the behaviors of the method under test(MUT) on the change of the states. There are two challenges in the generation of test cases for method-level unit-testing: 1. the generation of test inputs and expected outputs for a test case, and 2. the generation of the test fixture for a test case. The first challenge acquires approaches to partitioning the behaviors of the MUT into a more manageable collection of equivalence classes, to ensure that only a few representative test data are generated for each equivalence class. In our previous work [10], we transform OCL expressions into constraint logic graphs (CLG), in which, each complete path represents an equivalence class.

The second challenge is the test fixture computation problem. This problem states that the object states should be well initiated and compliant to the constraints specified in the software model to correctly exercise a test case. However, most of related works does not address the test fixture computation. A few related works that address the test fixture computation explore the space of the object states through the sequences of method invocations [2], [3]. Our previous work [10] didn't address the test fixture computation. This work extends our previous work to use the constraint-based approach to addressing the test fixture computation. This work uses relations, e.g., associations and multiplicities, defined in the UML class diagrams to generate test fixtures. The reasons are twofold: First, sequence of method invocations are relating the testing of the current method with other methods. The defects of other methods will reduce the adequacy of the generated test suites. Second, OCL expressions can specify relations among instances of classes, only the object states are well initiated and compliant to the relation constraints, can well perform a test data derived from OCL expressions.

The remainder of this paper is organized as follows: Section 2 describes the the modeling approaches adopted in this study; Section 3 describes the implementation details; Section 4 presents the performance evaluation; Section 5 provides a summary of related work; and finally, Section 6 provides our conclusions.

2. Generation of Test Cases

A test data is an abstract description to a test case, and is test-platform independent. A test case contains test data (test inputs and expected outputs) and test fixtures. The problem of generating test cases from the UML class diagrams and OCL can be modeled as a constraint satisfaction problem (CSP). The modeled CSP is composed by three sub-CSPs and they are, CSP_{PRE}, CSP_T and CSP_{POST}. CSP_{PRE} is the problem of finding the object states before performing the test. A solution to CSP_{PRE} is a test fixture candidate. CSP_{POST} is the problem of finding the object states after performing the test. CSP_T is the problem of generating test inputs and expected outputs from the equivalence classes of the constraints of the MUT. However, the involved pre/post object states of a solution of CSP_T should satisfy CSP_{PRE} and CSP_{POST} accordingly.

Consequently, the problem of test case generation can be modeled as follows. Let s_{pre} be a solution to CSP_{PRE}, s_{post} be a solution to CSP_{POST}, (a, r) be a solution to CSP_T, where a is the test inputs and r is the expected outputs, then a test case $t \equiv (s_{pre}, a, r, s_{post})$.

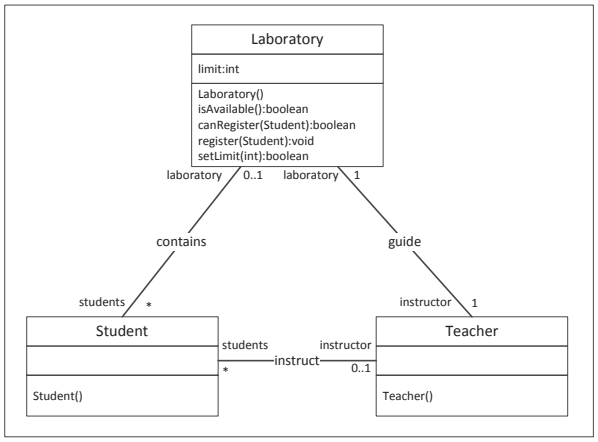


Fig. 1: An example UML diagram.

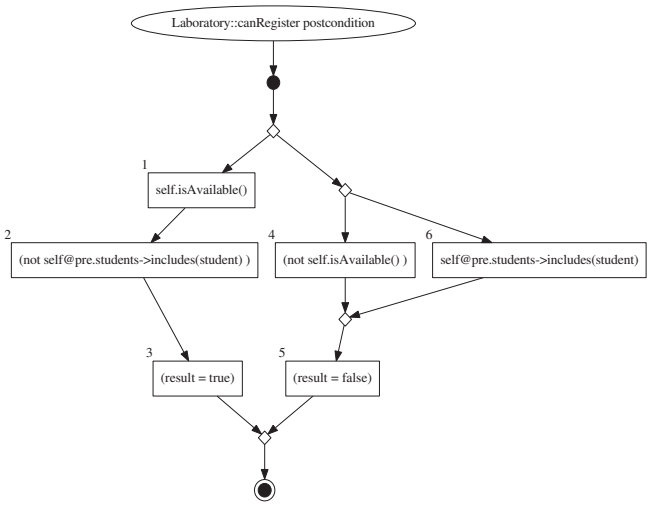


Fig. 2: The CLG of canRegister() method.

2.1 Generation of Test Fixture

The problems of CSP_{PRE} and CSP_{POST} are to find object states that are compliant to the constraints specified by the software model. The constraints describe the object states are defined in UML class diagrams and class invariant constraints of OCL specifications. An UML class diagram defines the attributes and methods of the classes, relations of associations and multiplicities among the classes. OCL class invariants of a class specify the constraints that must be held for each object of the class. In this work, we adopt the finite model reasoning approach [5] to model the problems of CSP_{PRE} and CSP_{POST}.

The finite model reasoning approach models the problem of validating an UML class diagram as a problem of finding a finite initialization of the UML class diagram. If an UML class diagram can find a finite number of objects and associations, which can be initialized according to the definitions and relations defined in the UML class diagram, this UML class diagram is said to be finite reasonable and to be a valid model. Consequently, we model the problems of CSP_{PRE} and CSP_{POST} in similar ways. If CSP_{PRE} is finite reasonable and s_{pre} is one of the solutions, s_{pre} is a test fixture candidate. If CSP_{POST} is finite reasonable and s_{post} is one of the solutions, s_{post} is a candidate of object states after performing the test.

2.2 Generation of Test Data

The method under test (MUT) is defined by a set of OCL precondition and postcondition constraints. These constraints specify the behaviors of the MUT. In this work, we leverage our previous work to transform the OCL pre/post constraints of the MUT into a constraint logic graph for the generation of equivalent classes of test data[10]. A constraint logic graph is a graphical representation of a disjunctive normal form of logic constraints.

```
1  context Laboratory
2  inv:
3    self.limit <= 3 and self.limit > 0 and self.students
4    ->size() >= 0 and self.students
5    ->size() <= self.limit and self.students
6    ->iterate(student : Student; acc : Boolean = true | acc and
         student.instructor = self.instructor)
7  context Laboratory::isAvailable() : Boolean
8  post:
9    result = self.students@pre->iterate(s:Student; r:Integer = 0 | r
         + 1) < self.limit@pre
10 context Laboratory::canRegister(student : Student) : Boolean
11 post:
12   if (self.isAvailable() and (not self.students@pre->includes(
         student))) then
13     result = true
14   else
15     result = false
16   endif
```

Listing 1: A portion of the OCL specifications.

A constraint logic graph (CLG) is a directed graph with seven types of nodes: start node, end node, constraint node, connection node, iterate-start node, iterate-end node and iterate-conjunction node. The constraint of a complete path, starts from the start node and ends in the end node, represents a conjunctive clause of the disjunctive normal form and corresponds to an equivalence class of test data of the MUT. Figure 2 is the constraint logic graph of method canRegister() of class Laboratory as defined in the example software specifications of Figure 1 and Listing 1. The complete path of {1, 2, 3} represents the constraints of a test data generation candidate. The constraints corresponding to this path is presented as follows:

$$\begin{gathered} self.isAvailable() \wedge \\ (not self.students- > includes(student)) \wedge \\ (result = true). \end{gathered} \quad (1)$$

3. System Implementation

A test case is the solution of a test case predicate. A test case predicate is composed by several predicates: instance initialization predicates, path predicates and method emulation predicates. The instance initialization predicates are

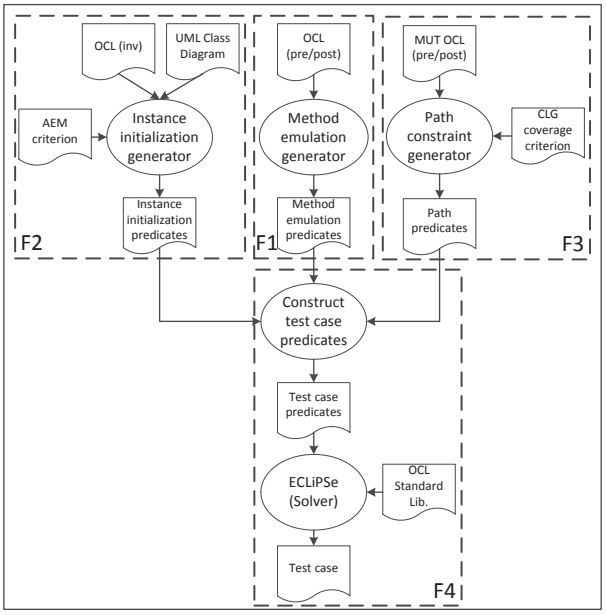


Fig. 3: The architecture of test case generator

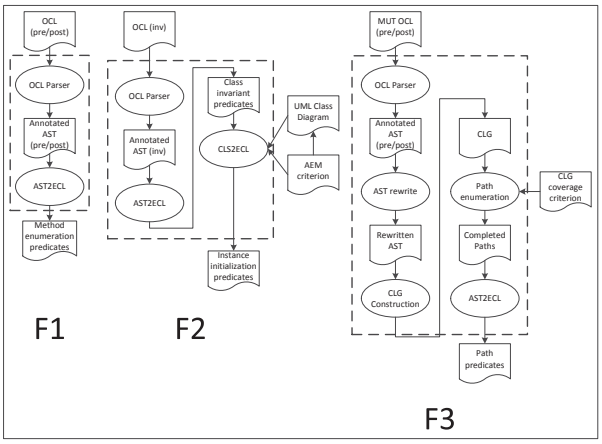


Fig. 4: Inner of each function group.

the representations of the test fixture computation problems of CSP_{PRE} and CSP_{POST}. The path predicates represent the constraints of a complete path of a CLG. Each method emulation predicate represents the pre/post constraints of a method of a class defined in the UML class diagram. Additionally, to support the predefined OCL types and operations, there is a set of predicates that implement the functions of the OCL Standard Library.

The architecture of the test case generator is depicted in Figure 3. This implementation is divided into four functional groups. The $F1$ functional group generates method emulation predicates from the OCL pre/post constraints of each method of the classes defined in the UML class diagram. The $F2$ functional group generates instance initialization predicates from the UML class diagram and class invariant constraints. An optional test coverage for class diagram provides additional coverage hints for the construction of instance initialization predicates. The $F3$ functional group generates path predicates from the OCL pre/post constraints of the MUT. The path enumeration is guided by a coverage criterion for the CLG. The $F4$ functional group constructs the test case predicates from the output of functional groups of $F1$, $F2$ and $F3$. The generated test case predicates, combined with the predicates of OCL Standard Library, are submitted to ECLiPSe[1] to find solutions. The solutions are test cases.

Table 1: Test data of constraint (1)

PRE	[(uml_obj, Teacher, 1)] [(uml_obj, Laboratory, 1, 1)] [(uml_obj, Student, 1)] [(uml_asc, guide, 1, 1)]
ARG	1 [(uml_obj, Laboratory, 1, 1), (uml_obj, Laboratory, 1, 1)] [(uml_obj, Student, 1), (uml_obj, Student, 1)]
POST	[(uml_obj, Teacher, 1)] [(uml_obj, Laboratory, 1, 1)] [(uml_obj, Student, 1)] [(uml_asc, guide, 1, 1)]

Table 1 is the generated test data for the constraint of (1). PRE and $POST$ are the solutions for problems of CSP_{PRE} and CSP_{POST}. A record with the mark of uml_obj is an object and with the mark of uml_asc is an instance of an association. The third element of each object records is the object identity. The third and fourth elements of an instance of an association are the participated object identities. ARG reports the test inputs and expected outputs of a test data. It is composed by three records. The first record is the expected output or the return value of the MUT, the second record is the pre/post states of the receiving object before and after the invocation of the MUT, the third record contains the inputs or the argument list to the MUT, the objects/variables in the argument list are also reported with their pre/post states.

3.1 Generation of Method Emulation Predicates

A method emulation predicate is a predicate that represents the conjunction of the pre/post constraints of a method. The procedure to generate method emulation predicates are depicted in $F1$ of Figure 4. Each of the pre/post constraints of the method is translated into an annotated AST, which is annotated with proper pre/post state information. AST2ECL translates ASTs into CLP predicates. A method emulation predicate is a conjunction of the generated CLP predicates.

We use the method isAvailable() of class Laboratory to demonstrate the generation process of a method emulation

[1]ECLiPSe is a constraint logic programming solution. It provides a set of tools and environment for constraint logic programming.

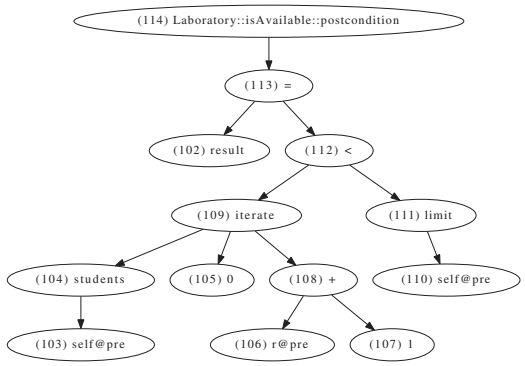


Fig. 5: The AST of method isAvailable().

predicate. Method isAvailabe() is defined by one postcondition constraint, and the associated AST is shown in Figure 5. The AST is visited in post-order and predicates for each visited nodes are generated. Listing 2 shows a portion of the generated predicates.

```
1  n_102_variable_result(_, Vars, Result) :-
2    ocl_variable(Vars, 2, Variable),
3    variable_state("postcondition", Variable, Result).
4  n_113_ocl_boolean_equals(Instances, Vars, Result) :-
5    Result = [ResultPre, ResultPost],
6    n_102_variable_result(Instances, Vars, [XPre, XPost]),
7    n_112_ocl_integer_less_than(Instances, Vars, [YPre, YPost]),
8    ocl_boolean_equals(_, _, XPre, YPre, ResultPre),
9    ocl_boolean_equals(_, _, XPost, YPost, ResultPost).
10 n_114_Laboratory_isAvailable_postcondition(Instances, Vars, Result)
       :-
11   n_113_ocl_boolean_equals(Instances, Vars, Result).
12 method_body_Laboratory_isAvailable(Instances, Vars, Result) :-
13   append(Vars, [Result|_], NewVars),
14   n_114_Laboratory_isAvailable_postcondition(Instances, NewVars,
         [1, 1]).
```

Listing 2: Predicates for emulating the method of isAvailable()

$Instances$, $Vars$ and $Result$ are three important variables. The $Instance$ variable contains the variables representing the instances of objects and associations. The details about the construction of the $Instance$ variable is described in Section 3.2. The $Result$ variable is the variable that reports the result of the predicate. The variable of $Vars$ in each predicate is responsible for propagating global and auto variables between predicates. The $Vars$ is a list of variables, which is defined as follows,

$$Vars \equiv [Self, Arg_1, Arg_2, \cdots, Arg_n, RetVal, ExVars], \quad (2)$$

where, $Self$ represents the receiving object under test, Arg_x are the variables that associate with the arguments of the MUT, $RetVal$ is the return value of the MUT, and $ExVars$ stores the auto variables propagated between predicates.

3.2 Generation of Instance Initialization Predicates

The construction of the instance initialization predicate is referring to the work [4], [5]. Listing 3 is an illustration of a generated instance initialization predicate. This predicate is composed by three portions of predicate: the cardinality resolution predicates (line #3 to #13), the instance creation

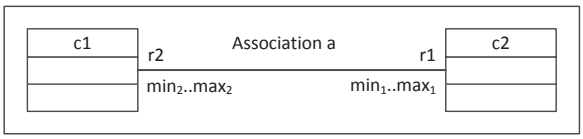


Fig. 6: An example binary association.

predicates (line #15 to #23), and class invariant predicates (line #25 to #29). The processes to generate instance initialization predicates are depicted in $F2$ of Figure 4.

```
1  createInstances(Instances):-
2      Instances = [OTeacher, OLaboratory, OStudent, Linstruct,
         Lcontains, Lguide],
3    %Cardinality definitions
4    ic:'::'(SLaboratory, 0..10),
5    % ...
6    ic:'::'(Scontains, 0..10),
7    % ...
8    CardVariables=[STeacher, SLaboratory, SStudent, Sinstruct,
       Scontains, Sguide],
9    %Adopt cadinality constraints
10   constraintscontainsCard(CardVariables),
11   % ...
12   %Instantiation of cardinality variables
13   ic:'labeling'(CardVariables),
14
15   %Object creation
16   creationLaboratory(OLaboratory, SLaboratory),
17   % ...
18   %Link creation
19   creationcontains(Lcontains, Scontains, Scontains, SStudent),
20   % ...
21   %Adopt multiplicity constraints
22   multiplicityLinkscontains(Instances),
23   % ...
24
25   %Adopt class invariants
26   (foreach(ILaboratory, OLaboratory), param(Instances) do(
27
28     n_34_Laboratory_invariant([Instances, Instances], [[ILaboratory
         , ILaboratory]], [1, 1])
29   )).
```

Listing 3: An instance initialization predicate

3.2.1 Generation of Cardinality Resolution Predicates

The cardinality resolution predicates declare a size variable for each class and association, e.g., $SLaboratory$ for class Laboratory, $Scontains$ for association contains. The size variable is to represent the possible number of initialized instances of the class/association. Each size variable is bounded by a given domain, to ensure the solver search for solutions in a finite range. The possible value of the size variables are also restricted by the cardinality constraints defined by the association relations. This implementation interprets only binary associations. Predicate constraintscontainsCard/1 in Listing 3 is one of the predicates specifying the cardinality constraints. Figure 6 is an example binary association, and the cardinality constraints are defined as follows [5],

$$\begin{aligned} N_a &\leq N_{c1} \times N_{c2} \\ min_1 \times N_{c1} &\leq N_a \leq max_1 \times N_{c1} \\ min_2 \times N_{c2} &\leq N_a \leq max_2 \times N_{c2}, \end{aligned} \quad (3)$$

where N_a, N_{c1} and N_{c2} are size variables for association a, class $c1$ and class $c2$. Constraint (3) describes the following

truth: 1. that the number of instance of association a is bounded by the multiplication of the number of the instance of class $c1$ and the number of the instance of class $c2$. 2. each instance of class $c1$ is associated with at least min_1 and at most max_1 instances of class $c2$ through the association a. 3. class $c2$ has the similar constraints.

3.2.2 Generation of Instance Creation Predicates

In the work of UMLtoCSP, a class is modeled as a struct of variables. Instances of the same class are given with unique object identities, Oid, to make distinction. In our implementation, we use list of variables to represent a class. This representation has better flexibility for matching during the stage of solving for answers. Taking class of Laboratory as an example, an instance of class Laboratory is a variable of list. The predicate to create instances of class Laboratory is defined in Listing 4.

```
1 creationLaboratory(Instances, Size):-
2   length(Instances, Size),
3   (foreach(Xi, Instances), param(Size) do
4   Xi = [uml_obj, "Laboratory", OidInteger, Integer1],
5   ic:'::'(OidInteger, 1..Size),
6   ic:'::'(Integer1, 0..5)).
```

Listing 4: Predicate for creating instances of class Laboratory

Associations are modeled in the similar approach, an instance of association contains is a list of variables. The predicate to create instances of association contains is listed in Listing 5.

```
1 creationcontains(Instances, Size, SStudent, SLaboratory):-
2   length(Instances, Size),
3   (foreach(Xi, Instances), param(SStudent), param(SLaboratory) do
4   Xi = [uml_asc, "contains", ValuePart1, ValuePart2],
5   ic:'#>'(ValuePart1, 0), ic:'#=<'(ValuePart1, SStudent),
6   ic:'#>'(ValuePart2, 0), ic:'#=<'(ValuePart2, SLaboratory)).
```

Listing 5: Predicate for creating instances of association contains

Multiplicity constraints must also be satisfied by each individual object of the participant classes. The predicate multiplicityLinkscontains/1 in Listing 3 is one of the multiplicity constraints. For the binary association of Figure 6, the multiplicity constraints are defined as the following FOL assertions [5]:

$$\begin{aligned} &\forall x.c_1(x) \rightarrow (min_1 \leq \sharp\{y|a(x,y)\} \leq max_1) \\ &\forall y.c_2(y) \rightarrow (min_2 \leq \sharp\{x|a(x,y)\} \leq max_2) \end{aligned} \quad (4)$$

where each instance of class $c1$ is associated with at least min_1 and at most max_1 instances of class $c2$ through the association a. Class $c2$ has the similar constraints.

3.2.3 Generation of Class Invariant Predicates

The generation of class invariant predicates is similar to the generation of method emulation predicates. Line #25 to #29 in Listing 3 are CLP codes to apply the generated class invariant predicates. In this example model, class Laboratory is the only class that have class invariant constraint.

3.3 Generation of Path Predicates

A path predicates represents the constraints of a complete path in the CLG of the MUT. The flow of generation of path predicates is depicted in $F3$ of Figure 4. A constraint logic graph is constructed from the ASTs of the pre/post constraints of the MUT. The details of the CLG construction can be found in our previous work [10]. The enumeration of paths from a CLG is guided by the selected coverage criterion. The constraints of each generated complete path represent an equivalence class that can be a candidate for test data generation.

The constraints of a complete path is a conjunction of the constraint expressions in the constraint nodes of the path. Each constraint expression is referring to an AST representation of the expression, and the associated predicate can be generated by AST2ECL. Consequently, a path predicate is a conjunction of the generated predicates of the constraint expressions. Taking the complete path of {1, 2, 3} of Figure 2 as an example, the generated path predicate is depicted in Listing 6 of line #11 to #13.

3.4 Generation of Test Case Predicates

A test case predicate is composed by instance initialization predicate for pre/post states and a path predicate. Listing 6 is an illustration of a test case predicate for the constraint (1). If a test data predicate has solution, the variable $InstancesPre$ will contain the object states (the test fixture) before performing the test, the variable $InstancePost$ will contain the object states after the test, and the $OutputVars$ will contain the test inputs, expected output and the receiving object to perform the test. Table 1 is a solution to the test case predicate of the constraint (1).

In Listing 6, line #7 the instances of pre-state are created, line #10 to #13 are the path predicates for path {1, 2, 3} of Figure 2 and line #15 is the instance initialization predicate for post-state.

```
1  tcgen_2_LaboratorycanRegister(InstancesPre, OutputVars,
       InstancesPost) :-
2    % ...
3    OutputVars = [Result, OLaboratory0, OStudent1],
4    Instances = [InstancesPre, InstancesPost],
5
6    % instance initialization predicate for pre-state
7    createInstances(InstancesPre),
8
9    % ...
10   % the path predicates for the equivalent class of path {1, 2, 3}
11   n_541_Laboratory_isAvailable(Instances, Vars, [1, 1]),
12   n_539_ocl_boolean_not(Instances, Vars, [1, 1]),
13   n_545_ocl_boolean_equals(Instances, Vars, [1, 1]),
14
15   % instance initialization predicate for post-state
16   createInstances(InstancesPost),
17   % ...
```

Listing 6: Test case generation predicate of path {1, 2, 3}.

3.5 Support of Test Coverage Criteria

The support of possible test coverage criteria are implementation dependent. Our implementation has direct support on structural based coverage criteria [11] and class diagram based coverage criteria [9]. The structural coverage

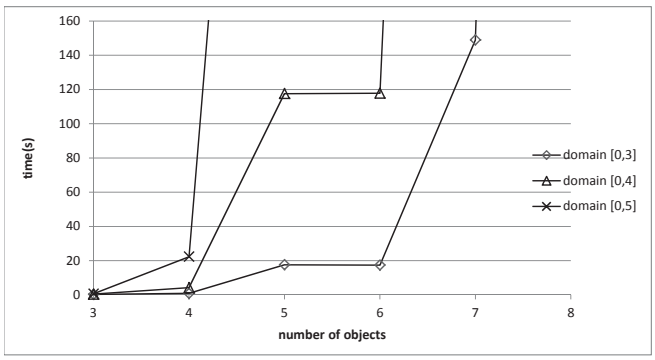


Fig. 7: Cost on different domains and minimum objects.

criteria are adopted on the generated CLG. For performance evaluation, we currently implemented structural coverage criteria of statement coverage (SC) and branch coverage (BC); and a class diagram coverage criterion of association-end multiplicity (AEM) [9]. However, more testing criteria can be supported in the future.

4. Performance and Quality Evaluation

In this section we perform several experiments to evaluate the performance of our implementation. We also adopt mutation testing to evaluate the quality of the generated test cases.

4.1 Performance Evaluation

To evaluate the performance of our implementation, we create several software models with different characteristics for the experiment. The characteristics of each model [2] and the time to generate test cases are summarized in Table 2. In this table, the info column shows information about the number of classes, associations and methods of each model; the clg column shows the total number of nodes and edges of the generated CLGs; the coverage column shows the selected coverage criteria.

Table 2: The cost to generate test data for different models.

model	info	clg	coverage	tests	time
Triangle	1/0/2	49/51	BC	5	331 ms
Date	1/0/2	196/224	BC	47	1786 ms
Laboratory	3/3/5	45/43	BC,AEM	27	4751 ms

According to the experiment results, this implementation can generate test cases for models of single class efficiently. Taking the model of Date as an example, this implementation generates 47 test cases in 1786 ms. The cost is far smaller comparing with test cases crafted by hand. However, for models of multiple classes and relations, the cost is increased.

[2]The UML class diagrams, OCL specifications, AST trees and CLGs of the listed models are available in the following URL for reference. http://zeus.cs.ccu.edu.tw/tcgen/

Name	Line Coverage		Mutation Coverage	
Date.java	96%	49/51	100%	45/45
Laboratory.java	80%	20/25	80%	12/15
Student.java	56%	5/9	100%	2/2
Teacher.java	56%	5/9	100%	2/2
Triangle.java	77%	17/22	100%	9/9

Fig. 8: The result of PIT against the generated test data.

Figure 7 shows the cost to generate test cases for the canRegister() method by giving restrictions to the minimum number of participated objects in each test case, and different values of the domains for classes and associations. According to the experiment results, the domain of the classes and associations are the most significant factors that affect the cost on generating test cases. Since the number of the involved objects are generally small for a test case of method-level unit-testing, our implementation is capable to generate test cases efficiently by giving small values for the domains of the classes and associations.

4.2 Quality Evaluation

To evaluate the quality of the generated test cases, we translate the test cases into test cases of JUnit test-platform for a set of java implementation of the models. We use the mutation testing tool, PIT [12], to test the quality of the generated test data. The result is showed in Figure 8. From the result, we observe that the quality of a test case generator depends heavily on the information that the generator can obtain. Classes of Date and Triangle are two classes that are modeled in detail in their OCL specifications. Thus, even the line coverage is not high, but the generated test cases are strong against mutation testing. We also find that the generated test cases for Laboratory are not good both in line coverage and mutation coverage. The major reason of the low score is the implementation of Laboratory contains some methods, e.g., getter and setter methods, that are not modeled in the software model. The generated test cases wont help on the non-modeled methods. However, if we exclude the survived mutation tests on getter and setter methods, we will get a score of 93% (14/15).

5. Related Work

In this section, we review related work that applies constraint-based approach to testing and studies that discuss the problem of test fixture computation for unit testing.

5.1 Preamble Computation for Unit-Testing

Engels, Guldali and Lohmann, [3] proposed a three steps approach to generate system states for unit testing. In the first step, input parameters are generated randomly for the operation to be tested. In the second step, pre/post constraints are adopted to compute the object structure containing the

operation with parameters from the first step. The third step is to construct a system state graph that represents the possible system states generated from operations invocations. By adopting search over the graph of the system states, the object structure computed in the second step can be eventually found. The path to the object structure in the system state graph represents a sequence of operation invocation that can lead to the system states to perform the test. However, if the sequence cannot be found, the search process will start over from the first step.

Colin, Legeard, and Peureux [2] partition each operation of the specification by determining the different effects of the operations and the effects are called effect predicates. Boundary goals are computed on the effect predicates by minimization and maximization metrics based on boundary coverage criteria chosen by the tester. This results in one or several minimum and maximum boundary goals for each effect predicate. Each boundary goal initiates corresponded boundary states by using fixture computation. The fixture computation is a process of discovery of the system states. The search process can be forward searching or backward searching.

5.2 Constraint-Based Testing

Constraint-based testing (CBT) is commonly adopted in different testing methodologies. Gotlieb, Botella, and Rueher, [6] proposed a method that first transforms the C program under test into static single assignment form so that each variable in the program is assigned statically at most once. This method then derives a system of constraints for different execution paths in the control flow graph using symbolic execution. Finally, this method uses dedicated constraint solvers to automatically generate the test input for each feasible execution path.

Meudec [7] used symbolic execution to collect a system of constraints for different execution paths in Ada programs. He then used constraint logic programming to automatically generate the test input for each feasible execution path. Dick and Faivre, presented a technique for automatic test case generation from the VDM specification [8]. They symbolically transformed the VDM specification for the program under test into a first-order predicate calculus in disjunctive normal form. Each conjunctive clause in the disjunctive normal form corresponds to an equivalence class. They also considered the generation of a sequence of method calls to bring the program into an appropriate system state for testing.

6. Conclusions

In this work, we implement a constraint-based test case generator for method-level unit-testing. Test cases can be generated from UML class diagrams and OCL specifications. The object states to exercise the generated test cases can also be generated uniformly. Our work demonstrates the possibility of generating test cases for unit-testing from interrelated software models. According to the performance evaluation, our implementation can generate test cases for method-level unit-testing efficiently. In this work, we also convert test cases into test scripts of JUnit test-platform to evaluate the quality of the test data by using mutation testing tools. The results show that test cases generated from properly modeled methods have good mutation coverage.

References

[1] R. A. Demilli and A. J. Offutt, Constraint–based Automatic test data generation, *IEEE Transcations on Software Engineering*, 1991, 17(9):900–910.

[2] S. Colin, B. Legeard and F. Peureux, Preamble computation in automated test case generation using constraint logic programming, *Journal of Software Testing, Verification and Reliability*, 2004, 14(3):213–235.

[3] G. Engels, B. Guldali and M. Lohmann, Towards model–driven unit testing, In Models in Software Engineering,Springer Berlin Heidelberg, 2007, pp. 182–192.

[4] J. Cabot, R. Clariso and D. Riera, Verification of UML/OCL class diagrams using constraint programming, In *Software Testing Verification and Validation Workshop*, April 2008, pp. 73–80.

[5] M. Cadoli, D. Calvanese, G. De Giacomo and T. Mancini, Finite Model Reasoning on UML Class Diagrams via Constraint Programming, *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing* , 2007, pp. 36–47.

[6] A. Gotlieb, B. Botella and M. Rueher, Automatic test data generation using constraint solving techniques, In *Proc. of the 1998 ACM/SIGSOFT Symposium on Software Testing and Analysis*, March 1998, pp. 53–62.

[7] C. Meudec, ATGen: Automatic test data generation using constraint logic programming and symbolic execution, *Journal of Software Testing, Verification and Reliability*, 2001, 11(2):81–96.

[8] J. Dick and A. Faivre, Automating the generation and sequencing of test cases from model–based specifications, In *Proc. of the 1st International Symposium on Formal Methods Europe*, April 1993, pp. 268–284.

[9] A. Andrews, R. France, S. Ghosh and G. Craig, Test adequacy criteria for UML design models, *Software Testing, Verification and Reliability*, 2003, 13(2): 95–127.

[10] C. K. Chang and N. W. Lin, A constraint-based framework for test case generation in method-level black-box unit testing. To be appeared in *Journal of Information Science and Engineering*, paper number 140822 in http://journal.iis.sinica.edu.tw/jise_acceptedpapers.html.

[11] G. J. Myers, C. Sandler and T. Badgett, The art of software testing, John Wiley and Sons, 2011.

[12] H. Coles, Pit mutation testing, http://pittest.org/.

Taxonomy Dimensions of Complexity Metrics

Bouchaib Falah[1], Kenneth Magel[2]
[1]*Al Akhawayn University, Ifrane, Morocco,*
[2]*North Dakota State University, Fargo, ND, USA*
[1]B.Falah@aui.ma, [2]Kenneth.magel@ndsu.edu

Abstract

Over the last several years, software engineers have devoted a great effort to measuring the complexity of computer programs and many software metrics have been introduced. These metrics have been invented for the purpose of identifying and evaluating the characteristics of computer programs. But, most of them have been defined and then tested only in a limited environment. Scientists proposed a set of complexity metrics that address many principles of object oriented software production to enhance and improve software development and maintenance. The aim of this paper is to present taxonomy of complexity metrics that, separately, evaluate structural and dynamic characteristics of size, control flow, and data. While most invented metrics applied to only the method and class levels of complexity, our approach uses metrics on each of the three levels: class, method, and statement.

Keywords: Complexity Metrics; Software Testing, Effectiveness, Data flow, Data usage; Taxonomy; Cohesion.

1. Introduction

Measurement makes interesting characteristics of products more visible and understandable [1, 2]. Appropriate measurement can identify useful patterns present in the product being measured [3]. It makes aspects and products more visible and understandable to us, giving us a better understanding of relationships among activities and entities. Measurement is not only useful, but it is necessary. It is needed at least for assessing the status of our applications, projects, products, and systems. Measurement does not only help us to understand what is happening during the development and maintenance of our projects, but it also allows us to control the interaction between the components of our project and encourages us to improve our projects and products.

There are a multitude of computer program software metrics that have been developed since the pioneering work of Halstead [4]. There are also several taxonomies that have been used to describe these metrics.

Nowadays, software is expected to have an extended lifespan, which makes the evaluation of its complexity at the early stages critical in upcoming maintenance. Indeed, complexity is proportional to the evolution of software. Software metrics were introduced as tools that allow us to obtain an objective measurement of the complexity of software. Hence, enabling software engineering to assess and manage software complexity. Reducing software costs is one of the major concerns of software engineering which creates an increasing need for new methodologies and techniques to control those costs. Software complexity metrics can help us to do so. In this paper, we would provide taxonomy of complexity metrics that can be served in reducing software costs. These metrics are used on each of the three levels: class, method, and statement.

2. Related Work

Many metrics have been invented. Most of them have been defined and then tested only in a limited environment. The most commonly used metrics for software are the number of lines of source code LOC (a rough measure of size), and Cyclomatic complexity (a rough measure of control flow).

Halstead software science [4] metrics are other common object oriented metrics that are used in the coding phase. Maurice Halstead's approach relies on mathematical relationships among the number of variables. His metrics, or what are commonly referred to as 'software science' [4], were proposed as means of determining quantitative measures directly from the operators and operands in the program. Halstead metrics

are used during the development phase with the goal of assessing the code of the program. Halstead's metrics are at the statement level, although they can be aggregated to form method and class level metrics.
Chidamber and Kemerer [5] proposed a set of complexity metrics that address many principles of object oriented software production to enhance and improve software development and maintenance. However, their metrics applied to only the method and class levels of complexity. They were evaluated against a wide range of complexity metrics proposed by other software researchers and experienced object oriented software developers. When these metrics are evaluated, small experiments are done to determine whether or not the metrics are effective predictors of how much time would be required to perform some task, such as documentation or answering questions about the software. Results have been mixed. Nevertheless, industry has adopted these metrics and others because they are better than nothing.

In recent years, much attention has been directed toward reducing software cost. To this end, software engineers have attempted to find relationships between the characteristics of programs and the complexity of doing programming tasks or achieving desirable properties in the resulting product such as traceability or security. The aim has been to create measures of software complexity to guide efforts to reduce software costs.

Our work applies a comprehensive suite of complexity metrics that can solve the problem of maximizing the effectiveness of software testing

3. Software Complexity Metrics

This paper uses software complexity metrics for object-oriented applications. Metrics for code that is not object oriented are not discussed in this research paper.
A metric is a measurement. Any measurement can be a useful metric. There are several reasons to use metrics in measuring the complexity of software, for instance:

- Prediction: metrics form the basis of any method for predicting schedule, resource needs, performance or reliability.
- Evaluation: metrics form the basis of determining how well we have done.
- Targeting: metrics form the basis for deciding how much effort to assign to which part of a task.
- Prioritization: metrics can form the basis for deciding what to do next.

Several researchers have proposed a wide variety of software complexity metrics. Each metric examines only one characteristic of software. This characteristic is one of:

- Size: how large is the software.
- Control Flow: either how varied is the possible flow or how deeply nested is the possible flow or how long is the possible flow.
- Data Usage: either how many data items are defined in the software or how many data items are related or how many values an attribute's value depend upon.

3.1. Size Metrics

One of the basic measures of a system is its size. Measures of software size include length, functionality, and complexity.

The oldest and most widely used size metric is the lines of code. The lines of code are common object oriented metrics that are used in the coding phase. There are two major ways to count the lines of code depending on what we count: a physical line of code (LOC) and a logical line of code (LLOC). While the common definition of LOC is the count of lines in text of the program's source code including comment lines, LLOC is defined to be the number of statements.
For example: if we consider the following Java fragment code:

```
if (int i = 0; i < 4; i++)  x[i] = i + 1;     // this is a line of code example.
```

In this example: LOC = 1 and LLOC = 2.

Another common OO metrics that are used in the coding phase were provided by Halstead software science [4]. Halstead's approach is based on the assumption that a program should be viewed as an expression of language. Halstead believed that the complexities of languages are an essential part of the reasons a programmer might find complexity in the program code. Therefore, he bases his approach on the mathematical relationships among the number of variables, the complexity of the code and the type of programming language statements

Because our research is related to Object Oriented Java Application, we will adopt the Halstead metrics to calculate the number of operators that are contained in each statement of a Java code program, then we will extend this metric to compute the total and the maximum number of operators of all statements within each method, and furthermore, we will compute the total and the maximum number of operators in all methods within the class. That means that we will use the number of operators in all three levels: class, method, and statement.

3.2. Control Flow Metrics

Another object oriented metric that is used in coding phase is McCabe Cyclomatic metric [6, 7]. Thomas McCabe developed his complexity metric in 1976. His approach was based on the assumption that the complexity

of software is related to the number of control paths generated by the code [6]. In other words, the code complexity is determined based on the number of control paths created by the code. This means that, in order to compute a code complexity, the number of decisions (if/then/else) and control statements (do while, while, for) in the code are the sole criterion for this purpose and therefore must be determined. For example, a simple function with no conditionals has only one path; a function with two conditionals has two paths. This metric is based on the logic that programs with simple conditionals are more easy to understand and hence less complex. Those with multiple conditionals are harder to understand and hence, more difficult and complex.

The control flow graph, G, of any given program can be drawn. Each node of the graph G corresponds to a block of code and each arc corresponds to a branch of decision in the program. The McCabe cyclomatic metric- [8] of such graph can be defined as:

$CC(G) = E - N + 2P$ (1) where,

- E: is the number of edges of G.
- N: is the number of nodes of G.
- P: is the number of connected components.

The formula (1) can also be written as:

$CC(G) = D + 1$ (2) where,

- D: is the number of decisions inside of the code.

Even if this information supplies only a portion of the complex picture, McCabe [7] tried to extend his metric into an architectural design and developed a testing methodology that integrates the notion of design complexity with the testing requirement.

3.3. Data Metrics

Data complexity metrics car be divided in two different aspects: data flow and data usage. Data flow is the number of formal parameters of activities and the mappings between activities' data [9]. We will define Data usage for a statement to be the number of variable values used in that statement plus the number of variable assigned new values in that statement.

The development of test cases of many researchers was based on the program unit's variables. The emphasis of test cases was based on data and data flow or Data-Usage Path [10]. Chidamber and Kemerer metrics [5], also known as C&K metrics, were among the first family of related metrics that address many concerns of OO designers including relationships such as coupling, cohesion, inheritance, and class size [11]. The notion of cohesion and the various complexity metrics associated with the cohesion are also related to data variables. In OO, the most widely C&K metric used example, when cohesion is related to instance variables, is Lack of Cohesion in Methods (LOCM) [12, 13].

Chidamber and Kemerer proposed a set of metrics that cover not just the data aspect but also cover other different aspects.

The C&K metrics are computed for each class in an application. Most of the metrics are at the class level while a few are at the method level. Figure 1, for example, illustrates how the C&K metrics would be apportioned among taxonomy dimensions.

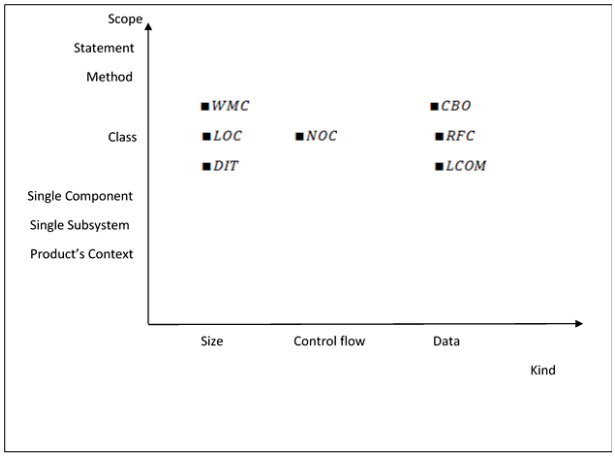


Figure 1. Taxonomy Dimensions of C&K Metrics.

While C&K metrics are used only at class and method levels, our approach uses metrics on each of the three levels: class, method, and statement.

Figure 2 illustrates how our suite of complexity metrics would be apportioned among our taxonomy dimensions.

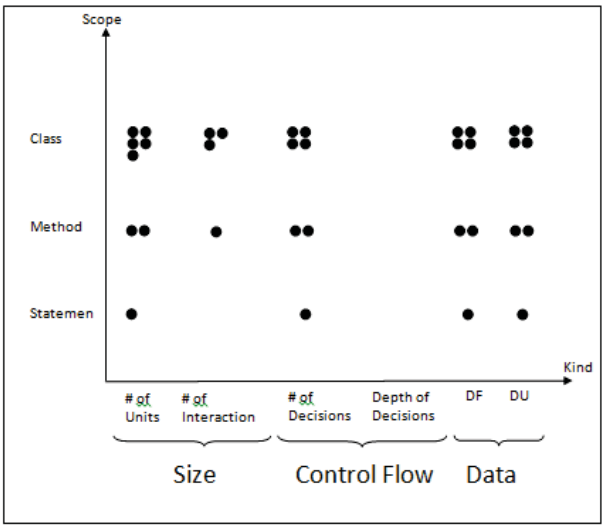


Figure 2. Taxonomy Dimensions of Our Approach.

4. Comprehensive Taxonomy of Metrics

Software engineers use measurement throughout the entire life cycle. Software developers measure the characteristics of software to get some sense of whether the requirements are consistent and complete, whether the design is of high quality, and whether the code is ready to be tested. Project Managers measure attributes of the product to be able to tell when the software will be ready for delivery and whether the budget will be exceeded. Customers measure aspects of the final product to determine if it meets the requirements and if its quality is sufficient. And maintainers must be able to assess and evaluate the product to see what should be upgraded and improved.

Software metrics usually are considered in one or two of four categories:

- Product: (e.g. lines of code)
- Process: (e.g. test cases produced)
- People: (e.g. inspections participated in)
- Value to the customer: (e.g. requirements completed)

In our work, we will concentrate on product metrics as selectors for test cases. Previous work using metrics almost always considered only a small set of metrics which measured only one or two aspects of product complexity.

Our work starts with the development of a comprehensive taxonomy of product metrics. We will base this taxonomy on two dimensions: (1) the level of the product to which the metric applies; and (2) the characteristic of product complexity that the metric measures.

In future work, we hope to produce a comprehensive taxonomy from the other kinds of metrics.
The scope of consideration dimension includes the following values:

(1) the product's context including other software and hardware with which the product interacts
(2) the entire product
(3) a single subsystem or layer
(4) a single component
(5) a class
(6) a method
(7) a statement

For the initial uses of this taxonomy reported in this paper, we will use only (5), (6), and (7) since they appear to be the most relevant scopes for unit testing. Future work may add (3) and (4) as we consider integration testing. Values (1) and (2) may be used for system testing.

The complexity kind dimension includes the following values:

1) Size
2) control flow
3) data

Each of these values in turn has sub-values.

For size, the sub-values are:

a) number of units (e.g. statements)
b) number of interactions (e.g. number of method calls)

For control flow, the sub-values are:

a) number of decisions
b) depth of decisions

For data, the sub-values are:

a) data usage
b) data flow

4.1. Metrics at Statement Level

4.1.1. Data Complexity. In our research, we consider two separate aspects, data flow and data usage. Data flow is based on the idea that changing the value of any variable will affect the values of the variables depending upon that variable's value. However, data usage is based on the number of data defined in the unit being considered or the number of data related to that unit. We will define data usage for a statement to be the number of variable values used in that statement plus the number of variable assigned new values in that statement.

Data flow complexity measures the structural complexity of the program. It measures the behavior of the data as it interacts with the program. It is a criteria that is based on the flow of data through the program. This criteria is developed to detect errors in data usage and concentrate on the interactions between variable definition and reference.

Several testers have chosen testing with data flow because data flow is closely related to Object Oriented cohesion [12, 14]. One measure of class cohesion is how methods are related through common data variables.

Data flow testing is a white box testing technique that can be used to detect inappropriate usage of data values due to coding errors [15]. For instance, a programmer might use a variable without defining it or might define a variable without initializing it (e.g. int a; if (a==1) {…}).

A program written in an OO language, such as Java, contains variables. Variables are defined by assigning values to them and are used in expressions. An assignment

statement such as: x = y + z ; defines the variable x. This statement also makes use of variables y and z. In this case, the variable x is called a definition while the variables y and z are called uses.

The declaration statement such as: int x, y, z; defines three variables x, y, and z. The three variables are assumed to be definitions.

In our research, data flow will be estimated for each statement in a method by counting how many active data values there are when the method executes. Active data values will be counted by determining which variable assignments could still be active when this statement begins execution plus the number of assignments done in this statement. As an example, let us consider the following Java class:

```
class DataFlowExample{
public void method1(int a, int b){
private int c = 0;
                                a = c + 3;
                                b = a + 1;
                                }
                        }
```

In the first statement of this code, the variable c is a definition. The same variable c is a use in the second statement. Thus, the data flow of this statement is 1.

In the second statement, a is a definition and c assigned a value. The variable a is a use in the third assignment. Thus the data flow value of the second statement is 2.

In the third statement, b is a definition and a assigns a new value. The variable b is no longer active before the method executes. Thus the data flow value of this third statement is 1.

On the other hand, as an example of data usage, let us consider the statement assignment: $x = y + z$.

The variables y and z are used, and the variable x is assigned a new value in the statement. Thus the data usage of this statement is 3.

4.1.2. Control Flow Complexity. In our research, we will use one control flow measure, the scope metric [16]. For each statement, we will count how many control constructs (do while, if-else, for, while …) contain this statement.

For example, assume that Figure 3 illustrates a statement fragment code of a return method named method C within the class "class C".

The construct level statements in this code are the statements numbered (6), (11), and (14).

```
1.      public class classC {
2.      public boolean methodC()
3.      {
4.              int x=0, y = 0, z = 0;
5.              boolean flag = false;
6.              while(flag)
7.              {
8.                      x = 2;
9.                      y = x + 1;
10.                     z = x + y;
11.                     if(y > 0)
12.                     {
13.                             y = y +1;
14.                             if(z == 0)
15.                                     z+= 2;
16.                             x+= 5;
17.                     }
18.             }
19.             return flag;
20.     }
21.     }
```

Figure 3. Java Code – Scope Metric Example.

Table 1 shows the scope metric value of each statement in the code of Figure 3.

Table 1. Scope Metric Values of Statements of Figure 5.

Statement	Construct Level contains the statement	Scope Metric Value
(4), (5)	None	0
(8), (9), (10)	(6)	1
(13)	(6), (11)	2
(15)	(6), (11), (14)	3
(16)	(6), (11)	2
(19)	None	0

4.1.3. Size Complexity. Our size metrics relied on the Halstead Software Science Definition. We will use a simplified version of Halstead's operators count discussed previously. Halstead's software science is one traditional code complexity measure that approaches the topic of code complexity from a unique perspective. Halstead counted traditional operators, such as + and ||, and punctuations, such as semicolon and ( ), where parentheses pair counted as just one single operator.

In our work, we will count just traditional operators for simplicity by counting the number of operators used in each statement of the code.

Figure 4 shows the metrics used in this research at the statement level. These four metrics will be used as roots to derive other complexity metrics that will be used at the method level and class level.

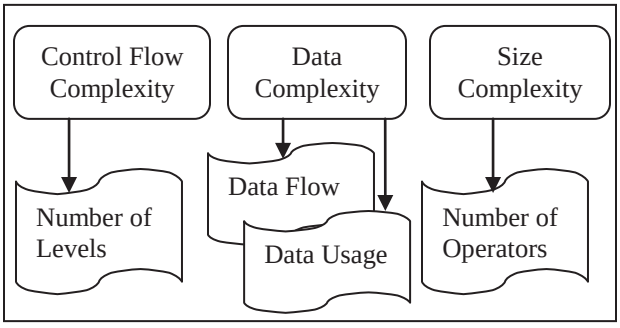


Figure 4. Complexity Perspectives and Metrics at Statement Level.

4.2. Metrics at Method Level

Since the method constitutes different statements, we will use both the sum of each metric for the statements within that method and the maximum value of each metric for the statements within that method. In addition to the sum and the maximum of these metrics, we will use another single level metric that counts the number of other methods within the same module (class) that call this method.

An example of this additional metric is shown in Figure 5.

```
public class ClassA {
        int[] arr = {1,2,3,5,6,9};
        public void method1()
        {
                for (int i=0; i<arr.length; i++)
                        System.out.println(arr[i]);
                method2(0);
        }
        public int method2(int y)
        {
                method1();
                for(int i=0; i<arr.length; i++)
                        y = y + arr[i];
                return y;
        }
        public void method3()
        {
                for(int i = 0; i< arr.length; i++)
                        arr[i] = arr[i] + method2(1);
                method1();
        }
}
```

Figure 5. Example of Other Methods that Call a Method within the Same Class.

For each method within the class "ClassA", the number of other methods that call that method with the same class is shown in Table 2.

Table 2. Metric Results of the Code in Figure 5.

Method	Other methods that call this method	Metric Value
method1	method2, method3	2
method2	method1, method3	2
method3	None	0

Figure 6 illustrates the nine metrics that will be used to measure the complexity of a method. Eight of these nine metrics are derived from the four metrics defined at statement level.

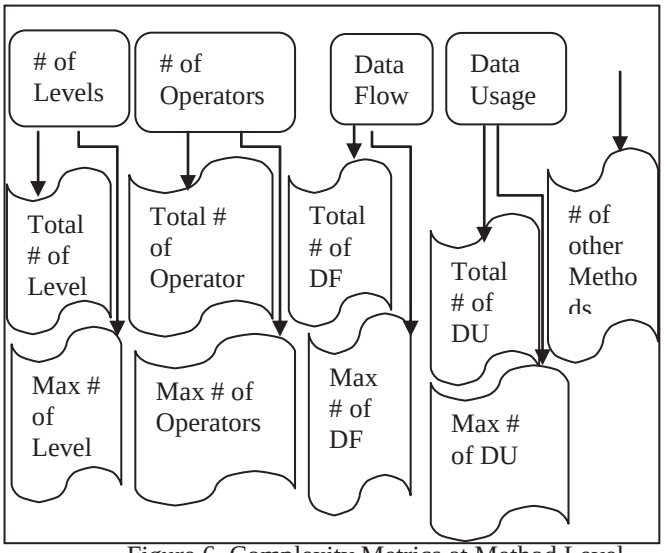


Figure 6. Complexity Metrics at Method Level.

4.3. Metrics at Class Level

At the class level, we will use both the sum of each metric for the methods within the class and the maximum value of each metric for the methods within the class. We will then add two additional metrics: the in-out degree of that class, which is the number of methods outside of that class that are called by at least one method in that class, and the number of public members within the class. The public members within a class are defined as the public fields and the public methods defined in that class.

As a summary of the comprehensive taxonomy of metrics that will be used in our research, for each executable statement within a method we will have 4 metrics that emerged from three complexity dimensions:

- Data Dimension: active data values and Data usage values.
- Control Dimension: scope metric.
- Size Dimension: number of operators.

For each method, we will have nine metrics. 2 metrics constitute the total and the max of the metrics of each statement within that method plus the number of other methods that call that method.
For each class, we will have twenty metrics, two metrics compose the total and the max of each of the 9 metrics that will be used for the method within that class, plus two more metrics including the number of methods outside of that class that are called by at least one method in that class, and the number of public members within the class.

5. Conclusion

This paper aims at developing a comprehensive taxonomy of product metrics that can be used to target test cases. This taxonomy is based on the metric dimension (product level) and the kind dimension (product complexity characteristic). We used the scope metric dimension values of class, method, and statement. We considered kind dimension values of size, control flow, and data. The three kind dimension values of product complexity have sub-categories. The size has the number of units and the number of interactions. The control flow has the number of decisions and the depth of decisions. The data has the data flow and the data usage.

In our work, we used at least one sub-category from each complexity kind dimension value. For the size, we used the number of units and the number of interactions. For the control flow, we used only number of decisions. For the data, we used data flow and data usage.

Another contribution of this research was the use of summation and maximum to build larger scope metrics from smaller scope metrics.

6. References

[1] B. Falah, K. Magel. "Test Case Selection Based on a Spectrum of Complexity Metrics". *Proceedings of 2012 on International Conference on Information Technology and Software Engineering (ITSE),* Lecture Notes in Electrical Engineering , Volume 212, 2013, pp. 223-235

[2] B. Falah, K. Magel, O. El Ariss. "A Complex Based Regression Test Selection Strategy", Computer Science & Engineering: *An International Journal (CSEIJ),* Vol.2, No.5, October 2012

[3] B. Falah. "*An Approach to Regression Test Selection Based on Complexity Metrics*" , Scholar's Press,
ISBN-10: 3639518683, ISBN-13: 978-3639518689, Pages: 136, October 28, 2013

[4] M.H. Halstead, "Elements of Software Science," Operating and programming systems series, New York: *Elsevier* North-Holland, 1977.

[5] S. R. Chidamber and C.F. Keremer, "A Metric Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No 6, June 1994, pages 476- 493

[6] T. J. McCabe and Charles Butler, "Design Complexity Measurement and Testing," *Communications of the ACM*, Vol. 32, Issue 12, December 1989.

[7] Thomas J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, No.4, December 1976.

[8] M. Clark, B. Salesky, C. Urmson, and D. Brenneman, "Measuring Software Complexity to Target Risky Modules in Autonomous Vehicle Systems," *AUVSI Unmanned Systems North America*, June 2008.

[9] J. Cardoso, "Control-Flow Complexity Measurement of Processes and Weyuker's Properties," *Word Academy of Science, Engineering and Technology,* August 2005.

[10] S. Rapps and E. Weyuker, "Selecting Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering*, Vol. SE- 11, No. 4, April 1985, pp. 367-375.

[11] R. Harrison, S. J. Counsell, and R.V. Nithi, "An Investigation into the Applicability and Validity of Object Oriented Design Metrics," *Empirical Software Engineering*, Vol. 3, Issue 3, September 1998.

[12] S. R. Chidamber and C.F. Keremer, "A Metric Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No 6, June 1994, pages 476- 493

[13] S.R. Chidamber and C. F. Kemerer, "Towards A Metrics Suite for Object Oriented Design," *In Proceeding of the ACM Conference on Object-Oriented Programming Systems*, Languages, and Applications. Vol. 26, Issue 11, November 1991.

[14] F. Damereu, "A Technique for Computer Detection and Correction of Spelling Errors," *Communications of the ACM*, Vol. 7, Issue 3, March 1964.

[15] J.P. Myers, "The Complexity of Software Testing," *Software Engineering Journal*, January 1992, pp. 13 – 24.

[16] H. F. Li and W. K. Cheung, "An Empirical Study of Software Metrics," *IEEE Transactions on Software Engineering*, Vol. 13, Issue 6, June 1987.

Quantitative Software Engineering

Igor Schagaev, London Metropolitan University, *i.schagaev@londonmet.ac.uk*
Svetlana Anulova, Ins of Problem Control, Russian Academy of Sciences
Hamid R. Arabnia, The University of Georgia, USA

Abstract

This paper proposes an analysis of a project model with feedbacks as palliative to qualitative models widely utilized in software engineering. It demonstrates that software engineering projects can reliably be modeled and analyzed quantitatively. Several analytical methods are introduced aiming to resolve problems of quantification. A sequence of steps to introduce rigor in software engineering models is explained showing how software warehouses can adopt proposed methods. Proposed approach when implemented can enhance the quality of produced software.

Keywords

Software engineering, qualitative models, feedbacks, quantitative modeling, simulation, semi-Markov models, analytic tractability.

1. Introduction

Software systems developed using ICT has become extremely complex and very often unmanageable. The problems of design and further maintenance have become a significant challenge for developers; in particular, in safety critical systems, including aviation and ground transport, pipelines, and others. The tangible cost associated with corrective measures is often extremely high.

For example, in recent years Chrysler [1],[2], Toyota [3], and other automobile manufacturers were forced to re-call hundreds of thousands of their new vehicles for corrective technological flaws that were not detected during manufacturing.

Due to various technological flaws, some corporations are even considering discontinuing some of their products; for example, Airbus is considering stopping the production of their A-380 [4]. There are many other examples of corporations discontinuing products solely due to costs associated with corrective measures and maintenance. It should be noted that a high percentage of cost of many products are due to software and electronics that operate the actual products. For example, 60% of cost of building aircrafts is due to ICT systems and the associated software.

Inherent in the design and creation of most systems and projects are User, Hardware, and Software. Obviously, any fault and flaws of one is intertwined with others. Thus projects with classic phases of concept, design, and development are becoming processes and phases of states that are tightly coupled in all steps of their development.

The challenge to control, manage, and streamline such processes have become an immense challenge. There are many potential solutions, based on methodologies, assumptions, methods to apply and follow but none of them considered to be solutions that would address the problem as a whole and estimate, let say, an impact of phases of the project on each other, values or weights of project phase dependencies or timing of decision making.

The software engineering researchers and community are publishing manuals [6], [7], new methodologies and ontologies (Agile, etc.) [8]. Many top tier international conferences, workshops, and symposiums report possible solutions every year. Unfortunately, most reported solutions, though important contributions, only address specific problems in isolation to other problems. Regretfully, there is no breakthrough on the subject. One of the major reasons is qualitative analysis and qualitative methodologies used.

This paper attempts to provide a scheme of redesign of software engineering models that in the long run would provide a quantitative measure. This paper also attempts to highlight problems of "quantification" and briefly present some solutions or pathways to find solutions.

2. Problem of uncertainty and qualitative approaches

The model of project process with feedbacks was introduced in [9] and first attempt of its quantification was introduced using simulation further developed in [10],[11]. Recent development [12] and two mentioned previous were based on various techniques of numerical solutions.

Project process as Markov process was introduced more than half a century ago by P. Howard [13], attempting to analyze project process analytically.

The costs of project phases were never certain, "generalization" of project phases or work packages independency failed dramatically even at the level of elementary projects, forcing us to accept that phase interdependencies are required to take into account. Therefore, managing our projects within budget and time constrains we have to handle project phases dependencies and handle them effectively.

Project complexity, primarily software development projects, growths and only increase the need develop some tangible solutions.

Indeed, having a quick look at the Figure 1 one might observe that phases of the project (states 1 to 3, and 4 is completion state) are dependent, and some redo takes place.

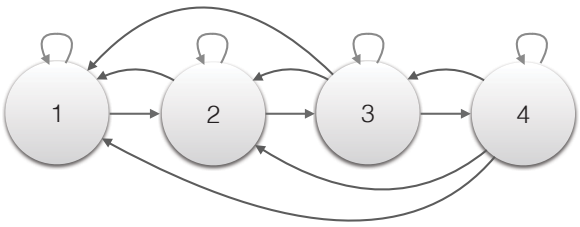


Figure 1 Project phases with full feedbacks

Immediate attempt to resolve this uncertainty was and still is based on formation of system of differential equations with assumption of Markov or semi-Markov properties of the states and transitions between them.

Regretfully, turning back to Fig.1 we can find that number of unknown variables (all feedback links from states 3,2,1 including self-feedbacks) exceeds number of equations, defined by number of project states. Thus the first option to solve it based on numerical solution, one of them is our own attempt, as it was presented in [10],[11]

Unfortunately, if we follow the same pathway, instead of solving project of process problems of improving efficiency, reliability and handling the cost of project we will be addressing mathematical complexities instead of project ones.

There is no doubt that analytical solution of process of project flow equations is preferable. This way we are able to determine dependencies between phases and learn how to cope with overheads of time or cost.

3. Existing and proposed solutions

There is no doubt, the less we use "guestimations", math methods assumptions to find a quantitative solution the better. We should avoid a situation when complexity of model exceeds complexity of a process we try to control.

Using [14], further developed [15],[16],[17] we might find that three redundancy types can be applied separately or in combination to reduce uncertainties existing in the description of the project as a model. They are:

Structural (S), Informational (I), Time (T).

Structural redundancy might be considered when we add new equations that reduce number of unknown variables.

Informational redundancy might be introduced as extra knowledge of behavior of the system of equations – for example number of iterations – i.e. how many times we have been visiting previous states, or profiles of values of feedbacks;

Time redundancy might be considered as introduced longer period of system observation.

Thus, introducing in the description of the system more equations, or more knowledge on behavior of unknown variables or observing this process longer enable us to find analytical solution.

One of the relatively recently developed models is presented in the next section.

4. Project as system with feedbacks and rewards dependent on past

R.A. Howard [13] introduced Markov chain with rewards to described economic objects and analysis of strategies for their control. Model became useful, a lot of research were taken, with introduction of generalization and special features.

Up to now there were no researches of Markov model when gain depends on the number of iterations of revisit projects states, and analysis of a reward scheme. Indeed, we are not redoing fully the project phase (states and Fig.1), but we simply must accept that further revisits might be reduced in accompanied cost.

This scheme suites well to software development project life cycle with feedbacks, where we assume that we are re-doing some bits of previous phases of project when errors or incompleteness of previous phases were detected.

The same models became useful for wider domains of human activities, including systems that include information and computer technologies, and in particular large-scale software projects.

Naturally, modeling of re-doing some steps of the project – should be rather typical case in everyday project practice. Surprisingly, this kind of model was not developed and missing so far.

Thus, as a first attempt we have to consider processes when gain depends on number of iterations of each phase of the project, perhaps with assumption of some sort of discounting factors.

Next section presents one of possible method of calculation of the project result (we use term reward to indicate that project cost might be too high) with assumptions of counting number of iterations and discounting of reward or cost of redoing.

Problem

Consider Markov chain with states *{0,…,K}* and transition *{ P = p_{ij}, i, j = 0,…, K}* and absorbing state *K*.

State of Markov chain at the moment $n = 0,1,\ldots$ denote *x(n)*.

Set trajectory x = (x(0), x(1),…) for the state $i \in \{0,\ldots,K\}$.

Denote as $\varphi_i(m)$ number of visits of the state *i* during period [0, 1,…, m], then

$$\phi_i(m) = \sum_{n=0}^{m} I_i(x(n)) \text{, and}$$

$$\phi(m) = \left(\phi_1(m),\ldots,\phi_K(m)\right)$$

For $N = 0, 1,\ldots, \infty$ consider random value

$$\xi(N) = \sum_{n=0}^{N} f(x(n), \phi(n)) ,$$

where

$$f:\{0, \ldots, K\} \times \{0, 1, \ldots\}^K \to [0, \infty), \quad f(K, \cdot) \equiv 0,$$

for example

$$f(i,\phi) = \frac{K-i}{1+\max\limits_{j\neq i}\phi_i} .$$

We attempt to derive $\mathsf{E}_i\xi(N)$, including $\mathsf{E}_i\xi(\infty)$ First impression that this scheme is similar to discrete Markov chain with reward [18]. But this is not.

New approach to calculation of $\xi(N)$ assumes extension of phase space --- $\{0, \ldots, K\}$ and, therefore process $\boldsymbol{x}$ in a way that enables to create Markov chain and function $\boldsymbol{f}$ as a fucntion of its own state only.

Then it becomes possible to apply recurrent scheme of first segment of Chapter 2 [13], travelling in simplex

$$\{\phi \in [0, 1, \ldots,)^K : \sum_{i=1}^{K}\phi_i \leq N\}$$

along the layers: from set

$$\left\{ \sum_{i=1}^{K}\phi_i = n \right\}$$

to set

$$\left\{ \sum_{i=1}^{K}\phi_i = n-1 \right\}$$

(These layers play a role in our case that in classic problem of [13] plays time).

At the moment we are not concentrating on this similarity, because our goal is to investigate cases when mean time of request in the network (chain) is possible to calculate analytically.

Case I: discounting accordingly number of visits

If $f(i,\phi) = f(i)\,e^{-(c_i,\phi)}$

Where $\boldsymbol{n}_i$ - vector with non-negative coordinates in $(c_i(0), c_i(1)\ldots, c_i(K))$, then classic scheme of Markov chains with rewards suits [18].

This becomes obvious if we note that

$$(c_i,\phi(N)) = \sum_{n=0}^{N} c_i(x(n)).$$

Theorem 1. Function

$$v:\{0,1,\ldots,K\}\to[0,\infty),\ v(i) = \mathbf{E}_i\xi(\infty)$$

is defined by system of linear equiations

$$v(K) = 0,$$

$$v(i) = e^{-c(i)}\left(f(i) + \sum_{j\neq i} p_{ij}v(j)\right).$$

Theorem enables immediate generalisation for the case when reward consists of sum of rewards as above, for example,

$$\sum_{i=1}^{K}\sum_{n=1}^{N} f_i(x(n))\exp\left\{-\sum_{m=0}^{n} c_i(x(m))\right\}.$$

Case II Function of reward with two variables

Consider a simplified problem: reward that received in a state $i\in\{0,\ldots,K\}$, depends not on the whole vector ϕ, but only on ϕ_i – number of visist (iterations) of this state. This means that function f depends on smaller number of variables:

$$f:\{0,1,\ldots,K\}\times\{0,1,\ldots\}\to[0,\infty)\ .$$

The idea here is to exploit a property of linearity of $\xi(I)$ by f.

Let f differ to zero only at the state $i\in\{0,\ldots,K\}$, in other words $f(j,l) = 0$ at $j\neq i$. Then

$$\mathbf{E}_i\xi(N) = \sum_{n=0}^{N}\mathbf{P}_i(n\le\phi_t(N))f(i,n)$$

Value $\mathbf{P}_i(n\le\phi_t(N))$ depends only on transitional probabilities and when $N\to\infty$ goes to p_i^n, where p_i denotes probability of the fact that chain leaving node i, returns to this node once more time. To define rigorously this probability, we introduce for $i\in\{0,\ldots,K-1\}$ a random variable τ_t - a Markov moment of chain in the state i

$$\tau_i = \min\{n\in\{1,2,\ldots\}: x(n) = i\}\quad(\min\{\varnothing\} = \infty)$$

(moment of the first visit after 0 iteration for the state i).

If Markov chain at moment is not in the state $\boldsymbol{i}$, then τ_t coincident with moment of reaching the state $\boldsymbol{i}$; while when process is in the state) then τ_t is a moment of first return to the state i.

Let $p_i = \mathbf{P}_i(\tau_t\le\tau_K)$

Then,

$$\mathbf{E}_i\xi(\infty) = \sum_{n=0}^{\infty} p_i^n f(i,n)\,.$$

Further, for $j\in\{0,\ldots,K-1\}$

$$\mathbf{E}_j\xi(\infty) = \mathbf{P}_j(\tau_i\le\tau_K)\mathbf{E}_i\xi(\infty)\,.$$

Theorem 2. Let $i\in\{0,\ldots,K-1\}$ and function $v:\{0,\ldots,K\}\to[0,1]$ match system of linear equations

$$v(j) = \sum_{l=0,\ldots,K} p_{jl}v(l),\quad j\neq i,$$

$$v(i) = 1,\quad v(K) = 0.$$

Then

$$p_i = p_{ii} + \sum_{j=0,\dots,K} p_{ij} v_i(j).$$

Proof is based on the well-known fact: value $v(j)$ – is a probability of the achieving the state *i* from state *j* earlier that in state k,

$$j \in \{0, \dots, K-1\}.$$

These two examples illustrate that analytic solution of possible when we introduce extra information about system behavior – number of iterations and rate of discount assumed along the process.

5. Next steps

Analytic modeling of project flow offers two areas of further research: a) regarding mathematical approaches – what else is possible to apply making analytics and b) which way analytic results might be applied – i.e. how to use the results of this analysis?

5.1 In math

Another missing point in creation of Markov model for project process is behavior of feedbacks. Indeed, an assumption of neat execution of phase after phase for any project is unrealistic; we have to assume non-zero probabilities of transition backward to previous phases.

Semantic of feedbacks provides some helpful dynamic constrain that ease analytic solution. It based on assumption of value of feedbacks. Clear, that the last thing project manager wants is total redoing of all phases of the project from the final step.

In turn, our assumption on non-zero values of feedbacks such as self-feedback – redoing of the same phase, or redoing of some elements of previous phases is much more natural.

Thus we might introduce an extra information about behavior of feedbacks assuming that probability of longer feedbacks is smaller than short ones and defined by, for example, a Poisson distribution. This will keep further solutions within reach of analytic method.

This form of *information redundancy* - the known form of ratio of feedbacks might be really useful. The use of *time redundancy* to resolve uncertainty of feedback values in our case present much less value in finding of project parameters.

5.2 In application for SE

Application of either first introduced here model - with counted feedbacks or the second mentioned - with information redundancy fine-tuning of feedbacks as a function of their length raises the following questions:

- *How can we apply analytic methods of project evaluation in practice of software projects?*
- *How can we apply the model of a project integrated with other projects within company?*
- *Can we apply integrated model of several projects as a single entity, using it at the level of corporation?*

The sequence below (Fig.2) presents one of the possible approaches.

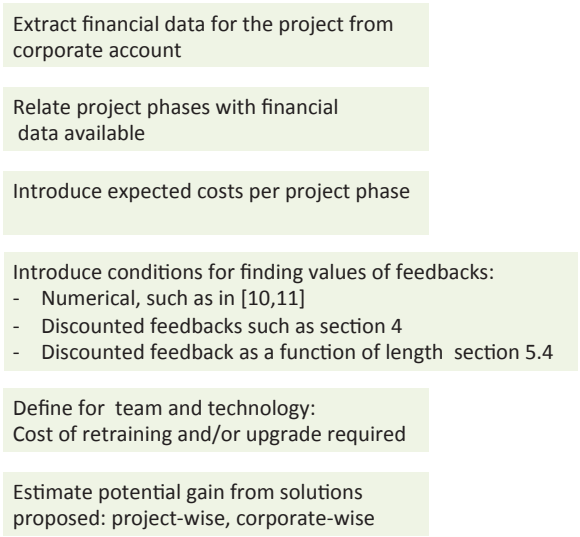


Figure 2 Implementation of QSE

At first, from existing financial data about project and project phases we have to extract the values of project phase planned cost and project phase.

Next we have to create a scheme, similar to Fig.1, introducing feedbacks between phases. Further – from corporate data generalized expected cost and time should be introduced.

Solving equations either with discounted returns or with introduction of distribution of feedbacks

as mentioned above we can estimate realistically project cost of concrete project and apply the scheme for similar projects.

For serious projects and corporate programs and state-size programs a customer or state representative (or - internally quality control analysts) might be able to provide values of feedbacks.

Having feedbacks values we can derive which distribution they obey. Having "golden" standard" for company data about projects – developers call it BEP – best existing practice our model enables to check level of competence and efficiency of project team in "the small" - per work package or independent task and "at large", taking into account dependencies of work packages and their impacts on each other.

We did apply this approach for international projects with respectable Seattle-based companies and several companies from various European countries. Very strange behavior of feedbacks was detected and this result was used as evidence of weaknesses of some teams or technologies applied along the phases of large-scale project.

Corporate-wise application is also possible, again by tuning a model for corporation or industry typical projects. Derived from this exercise pattern might helps to create an efficient working model. This way we might use a model to create a software tool to evaluate efficiency of corporate operations. Having this analysis a corporation might objectively assess when technology or competence of project teams is becoming obsolete and need upgrading.

6. Conclusion

- An attempt to introduce analytic schemes to quantify project process flow with realistic assumptions of overlapping phases is introduced.

- Shown that taking into account feedbacks between phases of the project makes model of the project realistic;
Ways to apply analytic solutions for the model of project with feedbacks are proposed and explained, resolving uncertainties along project progress.

- Further work of the quantification of software engineering and similar project control is possible in development of model with discounted feedbacks as function of their length;

- Proposed analysis and methods might become a core of application tools or software framework to enable project engineers and managers to analyze impact of their actions in advance.

References

[1]www.reuters.com/article/2014/10/16/us-chrysler-recalls-idUSKCN0I51D620141016
[2]http://money.cnn.com/2014/07/22/news/companies/chrysler-jeep-recall/
[3]http://www.washingtonpost.com/business/economy/toyota-reaches-12-billion-settlement-to-end-criminal-probe/2014/03/19/5738a3c4-af69-11e3-9627-c65021d6d572_story.html
[4] http://www.businessweek.com/articles/2014-12-11/the-double-decker-a380-faces-its-moment-of-truth
[5] http://www.globalresearch.ca/the-f-35-strike-fighter-technical-failures-of-the-worlds-most-expensive-weapons-system/5390065
[6] Pressman R., Bruce M., Software Engineering: A Practitioner's Approach, 8/e ISBN: 0078022126, 2015
[7] Sommerville I., Software Engineering, 9ed.,2010 Pearson
[8] Meyer B. Agile! The Good, the Hype and the Ugly. ISBN 978-3-319-05155-0
[9] Schagaev I. On Software Project Life Cycle, 1989 IAP Symposium, Savoy Place, London UK
[10] Pliaskota S., Schagaev I. Economic Efficiency of Fault Tolerance, Automatic and Remote Control, 1018-1026,vol 56., no 7, 1995.
[11] Pliaskota S., Schagaev I Life Cycle Economic Efficiency Analysis, Proc 2001 IEEE Systems, Man and Cybernetics Conf, Arizona, Tucson.
[12] arXiv: 1306.2365v2 [stat.ME] 28 Apr 2014
[13] Howard R.A., Dynamic Programming and Markov Processes. MIT Press, 1960
[14] Schagaev I. Yet Another Approach to Classification of Redundancy, PP485-491, 7th Symposium on Technical Diagnostic IMEKO,17-19 September, Helsinki, 1990
[15] Schagaev I., Reliability of malfunction tolerance. http://www.proceedings2008.imcsit.org/pliks/218.pdf
[16] Kaegi T., Schagaev I., System Software Support of Hardware Efficiency, IT-ACS Ltd, 2013, ISBN 978-0-9575049-0-5
[17] Castano V., Schagaev I. Resilient computer system design, Springer2015, ISBN 978-3-319-15068-0
[18] Mine H. and S. Osaki. (1970) Markovian Decision Processes. Modern Analytic and Computational Methods in Science and Mathematics, Elsevier, NY.

Debugging Multi-Threaded Applications using Pin-Augmented GDB (PGDB)

Nachiketa Chatterjee[1], Srijoni Majumdar[2], Shila Rani Sahoo[2], and Partha Pratim Das[3]
[1]A. K. Choudhury School of Information Technology,University of Calcutta, Kolkata, West Bengal, India
[2]School of Information Technology,Indian Institute of Technology, Kharagpur, West Bengal, India
[3]Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, West Bengal, India

Abstract— *In contrast to single threaded applications, debugging multi-threaded applications is complex because of the non-deterministic nature of concurrent programs. Multiple threads in concurrent programs introduce bugs like datarace, deadlock and livelock. Popular debuggers like GNU Debugger (GDB), Intel Debugger (IDB) and Microsoft Visual Studio Debugger (MVSD) typically use static or compile-time instrumentation and provide many features to debug single threaded programs. However the features dealing with debugging concurrency is limited. In this paper we explore dynamic instrumentation using JIT (Just-In-Time) compilation techniques for run-time behaviour using dynamic instrumentation framework from Intel PIN [1]. Using PIN we augment GDB with support for datarace and deadlock detection with automated breakpoint to GDB. We call it PGDB or PIN-augmented GDB - a multi-threaded debugging platform. We present here a prototype of PGDB for detecting datarace and deadlock during the execution of multi-threaded programs with the support of new commands in PGDB.*

Keywords: Multi-threaded debugging, datarace, deadlock,PGDB or PIN-augmented GDB

1. Introduction

Debugging multi-threaded applications is complex because of the non-deterministic nature of concurrent programs leading to concurrency issues like race conditions and deadlocks. *Datarace* occurs when two or more threads in a program access the same memory location concurrently without using any exclusive locks to serialize their accesses and with atleast one access for write. *Deadlock* is a condition in which two or more threads wait for each other to release a shared resource before resuming their execution.

The classical approach to debugging single threaded applications (sequential programs) involves repeatedly stopping the program, examining the state, and then either continuing or re-executing to stop at an earlier point in execution. Such debugging cycles help developers trace the sequential execution paths well but unfortunately do not identify the concurrency issues in multi-threaded programs.

Most popular debuggers like GNU Debugger (GDB) [4], Intel Debugger (IDB) [5] and Microsoft Visual Studio Debugger (MVSD) [6] provide many features to debug single-threaded programs. However the features dealing with debugging concurrency is limited (Table 1). Earlier Shi et al [7] used PIN to extract different threaded behaviours of applications by displaying the access / change history of a shared variable, tracking locks held by threads and displaying information at the breakpoints but datarace or deadlock was not detected. Also the method incurs huge overhead for instrumentation. In this background our objective is to augment the capabilities of existing debuggers with more multi-threaded support to help debug concurrency issues.

A debugger typically uses static[1] or compile-time instrumentation. However, we choose dynamic[2] or runtime instrumentation so that we can attach / detach debugging support on-the-fly without changing compiled code.For dynamic instrumentation we use the PIN [3] framework from Intel. Using PIN we augment GDB with support for datarace and deadlock detection with automated breakpoint to GDB. We call it PGDB or PIN-augmented GDB. Besides new GDB commands for datarace and deadlock, we also support an option to selectively enable/disable the detection mechanism to reduce the overhead of dynamic instrumentation during program execution.We have tested efficiency and accuracy of PGDB by developing benchmark test cases.

Though the design of PGDB is agnostic to the platform, the programming language or the multi-threading model, our implementation here is based on GDB on Linux with C/C++ language and pthreads [8] library for multi-threading support.

This paper is organized as follows. In Section 2 we outline the architecture of PGDB based on GDB, PIN and their interconnection. The instrumentation mechanisms to empower GDB detect concurrency issues like race condition and deadlock are presented in Section 3 and the implementation aspects covering new GDB commands for concurrency

[1]In *Static Instrumentation* the source code is instrumented during compilation and is used to identify the static program information.

[2]In *Dynamic Instrumentation* the binary (executable) code is instrumented using JIT (Just-In-Time) compilation to collect run-time information.

Table 1: Comparison of PGDB with Existing Debuggers

Feature/Debugger	PGDB	GDB [4]	IDB [5]	MVSD [6]
Examining state of existing threads	✓	✓	✓	✓
Thread specific breakpoints	✓	✓	✓	✓
Thread synchronizing breakpoints	✓	✓	✓	✓
Thread data sharing events	☑	×	✓	✓
Automatic notification of new threads	✓	✓	✓	✓
Logging Feature	☑	×	×	×
Replay Feature	×	×	×	×
Datarace detection	☑	×	×	×
Deadlock detection	☑	×	×	×
Livelock detection	×	×	×	×

☑ - Additional Feature ✓- Feature Present × -Feature Absent

support and enhancement of GDB GUI for user feedback are discussed in Section 4. We present a sample debugging session with PGDB in Section 5. The results debugging and detection for a set of benchmark codes, designed specifically to cover the corner cases of correctness of PGDB, are discussed in Section 6. We conclude in Section 7 with directions for future work.

2. Architecture of PGDB

To augment GDB with the intended multi-threaded debugging features we need the following primitives:

- *Control over memory accesses used by program* to identify memory instructions.
- *Identification of read and write accesses* to know the purpose (reading or writing) of memory access by an instruction.
- *Control over thread granularity* to find the thread ID executing a given instruction and to notify when a particular thread gets created or destroyed.
- *Control over routine granularity* to notify the start and completion of a routine[3].
- *Control over lock* to notify when the locks of a shared-exclusive memory are acquired or released.
- *Control over memory barrier* to identify the user defined synchronization using memory barrier.

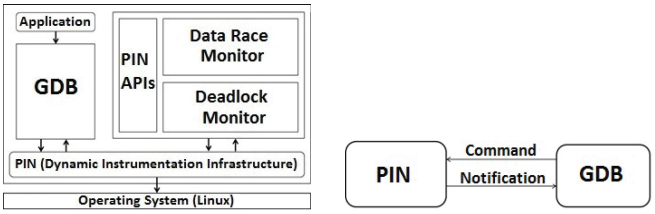
We use the dynamic instrumentation framework of PIN[4] to create pintools that extract the above primitives during the execution of an application under the control of GDB. Since we use pthreads the events of acquiring and releasing the locks are captured from the invocations of `pthread_mutex_lock()` and `pthread_mutex_unlock()` functions respectively. The resulting architecture of PGDB is shown in Figure 1 A. To use PGDB a developer needs to compile the source code in debug mode to create a GDB-compatible binary. The binary then executes in remote mode with PIN having custom instrumentations (as pintools). Finally, GDB is started and connects to PIN through remote port to debug and detect concurrency issues. The developer controls the debugging by issuing our new concurrency detection commands from GDB.

[3]A function is referred to as *routine* by PIN.

[4]PIN [1] is a binary instrumentation framework on Linux or Windows. A wide variety of program analysis tools, called Pintools [3], can be built using PIN. PIN is a JIT compiler that can inject instrumentation routines in instruction, basic block, routine or image level units. An instrumentation routine is attached as a callback either before or after an instrumentation unit. The design of these callbacks decide the behaviour of the pintool.

2.1 Interconnection of GDB with Pin

GDB supports a *remote* mode where it can communicate to the remote *stub* that understands GDB protocol via a Serial or TCP/IP connection. In PGDB PIN connects to GDB via its remote debugging protocol. The communication with the debugger is two-way as shown in Figure 1 B. GDB sends commands to PIN and PIN sends notifications back whenever the program stops on intended breakpoints or terminates. Note that every program instruction that is executed under the control of the debugger is still instrumented with the PIN.



(a) (b)

Fig. 1: (A) The Basic Architecture of PGDB (B) Interconnection of GDB with Pin via Remote Debugging Protocol

2.2 Breakpoint Propagation to GDB

PIN provides an API to generate breakpoint in GDB to stop during program execution. When the Data Race Monitor pintool identifies a potential Datarace or the Deadlock Detection pintool finds a deadlock event, a suitable breakpoint is generated from within the respective pintool and is passed on to the GDB console to stop the program execution. This carries the thread ID and message for GDB console.

3. Design of PGDB

In PGDB we augment GDB with features to detect the race condition and deadlock. These features may be turned on or off dynamically during the execution of an application under debug. We instrument `RecordLockBefore()` and `RecordLockAfter()` before and after the calls to `pthread_mutex_lock()` respectively. Further we instrument `RecordUnLockAfter()` after

calls to `pthread_mutex_unlock()`. Following instrumentations implement these features:

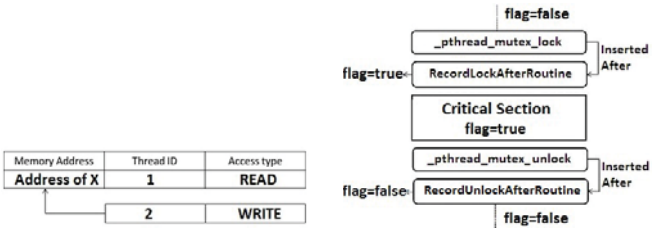
3.1 Data Race (Detection) Monitor

PIN provides APIs to identify memory accesses and to detect if the thread holds a lock on it or not during the access. Thus, to detect datarace we first identify the shared-exclusive memory locations and then monitor these locations for accesses with or without lock by a thread.

Instrumentation Policy

To identify whether or not a memory location is shared among multiple threads, we maintain a hash table `MemTracker` where the key-field is a (32- or 64-bit) memory address and the value-field is a memory region structure containing the Thread ID and the Access type (READ/WRITE). We perform image instrumentation to get the address range of the main executable image as loaded into memory and shared library images to filter out memory accesses for the main executable only as thread-local and read-only memory accesses do not induce dataraces.We instrument every instruction belonging to the main executable image and shared library images. Identifying memory locations is performed in two analysis routines namely `RecordMemRead()` and`RecordMemWrite()` before Load and Store instructions respectively. We use thread IDs assigned by PIN for thread identification within analysis routine

- **`RecordMemRead` Routine** is called before the execution of a *Load* instruction to analyse read accesses from memory. The memory address, thread ID and context are passed to this routine. When a memory address is accessed for the first time a memory region structure is populated with READ and is added to the `MemTracker`. For subsequent accesses for a memory location for which a memory region structure already exists in `MemTracker` we have the following situations:
  - **Existing Access type is READ**: This is a case of READ-after-READ and there is no datarace.
  - **Existing Access type is WRITE**: This is a case of READ-after-WRITE. There is no action (and no race) if the thread IDs are same. If the thread IDs are different, this memory location should be marked as a shared-exclusive memory.
- **`RecordMemWrite` Routine** is called before the execution of a *Store* instruction to analyse write accesses to memory. The memory address, thread ID and context are passed to this routine. When a memory address is accessed for the first time a memory region structure is populated with WRITE and is added to the `MemTracker`. For subsequent accesses for a memory location for which a memory region structure already exists in `MemTracker` we have the following situation:
  - **Existing Access type is READ or WRITE**: This is a case of WRITE-after-READ or WRITE-after-WRITE. Hence this memory location should be marked as a shared-exclusive memory if the thread IDs are different.



(a) (b)

Fig. 2: (A) Race Detection (B) Identification of Safe / Unsafe Access

We maintain a boolean variable `flag` (initialized to false) for each thread for race detection (Figure 2 B). When `RecordLockAfter()` is called, say, by Thread 1, we enter the critical section and set `flag` for Thread 1 as true. Later when `RecordUnlockAfter()` is called, we know that the thread has left the critical section and we reset the `flag` for Thread 1. Hence any access to a shared-exclusive memory location is a safe access while `flag` is true. Otherwise it is unsafe.A memory location if marked as shared-exclusive and has an unsafe access is a potential for datarace invoking the breakpoint.

The above characterization of safety, however, changes when users employ barriers for explicit synchronization. We enumerate different cases of safety with and without barriers in Table 2 and use them to formulate the following analysis strategy for exploring datarace in the presence of barriers.

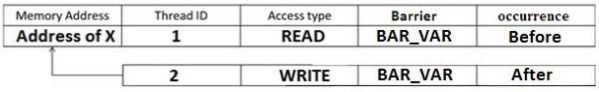
- **`BarrierDetect` Routine** is called before every call of `pthread_barrier_wait()` to track the memory barriers. The `MemTracker` is now extended with additional fields to store the barrier variable associated with a thread and the order of the occurrence (Before / After) of variables relative to the barrier. And if there is no barrier at all then these fields will be `NULL`.
  - **When we encounter a variable X in a thread before crossing a barrier**, we insert a new row in `Memtracker` as {<Mem_Addr>, <Thread_ID>, <Access_type>, <No>, <NULL>}. Now we have two possibilities either we cross a barrier after this variable or the barrier is not at all available.
    - **if variable occurs before a barrier** then during crossing the barrier named, say

BAR_VAR, the pthread_barrier_wait() callback will update the above row in Memtracker as {<Mem_Addr>, <Thread_ID>, <Access_type>, <BAR_VAR>, <Before>}.

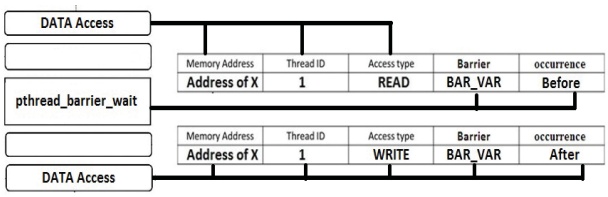
- ∗ **if there is no barrier at all** then the row in Memtracker will remain unchanged.

– **If we have already crossed a barrier**, say BAR_VAR, before accessing the variable then we insert {<Mem_Addr>, <Thread_ID>, <Access_type>, <BAR_VAR>, <After>} in Memtracker.

Here BAR_VAR will be the barrier variable name to deal with multiple barriers. If the memory access in two



(a)



(b)

Fig. 3: (A) No Race Due to Barrier (B) Updating Memtracker

threads appear before and after the barrier respectively (or vice-versa) like in Figure 3 A we can exclude this condition from potential data-race factors. Then we can formulate the additional logic for any variable X appearing in threads T1 and T2 before or after a barrier as cases 3, 4 and 5 of the Table 2.

```
If (((Occurrence of X in T1 is before barrier)
     AND
     (Occurrence of X in T2 is after barrier))
   OR
    ((Occurrence of X in T2 is before barrier)
     AND
     (Occurrence of X in T1 is after barrier)))
then NO DATA RACE.
```

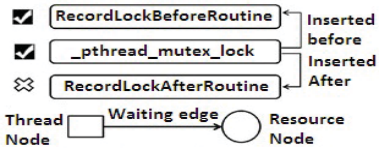
3.2 Deadlock (Detection) Monitor

To detect deadlock we use a *Resource Allocation Graph* (RAG). An RAG is a directed bipartite graph with two types of nodes and two types of edges. A RAG represents a thread by a *Thread Node* and a resource by a *Resource Node*. If a thread t owns (holds a lock on) a resource r, we draw an *Acquired Edge*(Figure 4 B) from r to t. If a thread t is blocked on a resource r, we draw a *Waiting Edge*(Figure 4 A) from t to r. Clearly, there is deadlock if and only if there is a cycle in the RAG. Thus, we can detect deadlock by building the RAG (Figure 4 C).

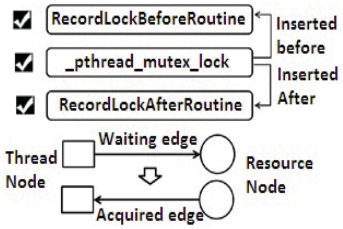
Instrumentation Policy

To construct a RAG we identify the waiting and acquired edges as follows:

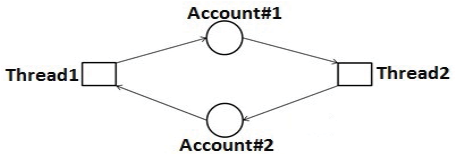
- A waiting edge from thread t to resource (mutex) r is added to the RAG when t is blocked in pthread_mutex_lock(&r) routine because some other thread holds the lock on the mutex r. This is done in RecordLockBefore().
- An acquired edge from thread t to resource (mutex) r is added to the RAG when t acquires the lock on the mutex r by completing pthread_mutex_lock(&r). While an acquired edge is added the existing waiting edge is removed. This is done in RecordLockAfter().
- An acquired edge is removed from the RAG when thread t releases mutex r by completing pthread_mutex_unlock(&r). This is done in RecordUnLockAfter().



(a)



(b)



(c)

Fig. 4: (A) Waiting edge in RAG (B) Acquired edge in RAG (C) Deadlock detection by RAG

Once the RAG is constructed we detect deadlocks by finding cycles in it and the deadlock breakpoint is invoked.

4. Implementation of PGDB

We have implemented the above design over GDB to create PGDB. For this we have added a set of new commands to GDB (Section 4.1). The users can use these command as input to PGDB to control the debugging of concurrency. Further we have enhanced the GUI mode of GDB (Section 4.2) to output debugging information back to the user.

4.1 PGDB Commands

To use the detection monitors from PGDB, we have designed the following simple commands that the developer can issue to PIN through PGDB:

- **monitor help**: Display all customized commands to debug concurrency issues.
- **datarace detection on**: Start the detection of shared-exclusive memory access (datarace).
- **datarace detection off**: Stop the detection of shared-exclusive memory access (datarace).
- **datarace detection status**: Shows whether the detection of datarace is ON or OFF.
- **deadlock detection on**: Start the detection of deadlock.
- **deadlock detection off**: Stop the detection of deadlock.
- **deadlock detection status**: Shows whether the detection of deadlock is ON or OFF.

4.2 Enhancement of GUI Mode of GDB

We have enhanced the support of graphical framework of GDB and built the infrastructure to generate breakpoints by our profilers. Among all existing graphical gdb in the market, we have chosen DDD for this purpose which is a well-known graphical gdb with clean and simple interface. Once DDD layer comes on top of our PGDB, whenever a breakpoint is generated by our profiler, it needs to be propagated to the GUI layer from GDB so as to highlight the cause of the detection by profiler in the source code of the application. This is done very interactively using a cursor pointing the exact line causing the detection of deadlock, datarace.

To integrate our new features with GDB, we designed a shell script to automatically establish the connection between PIN and GDB, and instantiate DDD irrespective of the system architecture on which it is running.Higher level steps are given below:

- Instantiate PIN with the designed Pintool or Profiler
- Start DDD interface which will run the GDB engine in the background
- Establish remote connection between GDB and PIN, as the application to be debugged will be running in PIN which is outside GDB
- Open a separate source window of DDD displaying the source code of test application
- Open a separate data window where variable values will be shown while debugging
- Open the GDB command window to input the custom commands and run the program

5. Sample Debugging Session in PGDB

We present example debugging in PGDB implementation.

5.1 Data Race Breakpoint Feature

Consider the following code being executed by Thread 1. Suppose Thread 2 increments shared variable `x` without a lock. Hence the value of `y` is not deterministic due to race.

```
if (x == 5) // The "Check"
{ y = x * 2;  // If another thread changed x
    // in between "if (x == 5)" and "y = x * 2";
    // y will not be equal to 10.}
```

PGDB identifies the race and halts the program with a breakpoint. When Thread 1 reads the value of `x`, it is registered as a READ operation in `MemTracker` and when Thread 2 increments its value, the race condition is detected (Figure 2 A).

5.2 Deadlock Detection Breakpoint Feature

An example of deadlock is shown below where two threads, Thread 1 and Thread 2, invoke `transfer()` as shown:

```
    void transfer(Account from_account,
        Account to_account, double amount) {
        pthread_mutex_lock(&from_account);
        pthread_mutex_lock(&to_account);

        from_account.withdraw(amount);
        to_account.deposit(amount);

        pthread_mutex_unlock(&to_account);
        pthread_mutex_unlock(&from_account);
    }
Thread 1: transfer(account#1, account#2, 1000);
Thread 2: transfer(account#2, account#1, 500);
```

As Thread 1 starts executing `transfer()` it holds lock on `account#1` and is suspended as it waits for lock on `account#2`. Meanwhile Thread 2 acquires lock on `account#2` and waits indefinitely for Thread 1 to release `account#1`. Deadlock results. A cycle in this RAG (Figure 4 C) implies deadlock.

6. Test Result

The benchmark test suite for test datarace and deadlock are demonstrated in Sections 6.1 and 6.2 respectively. The behaviour and performance on the benchmarks have been presented in Sections 6.3 and 6.4.

6.1 Correctness for Datarace

The following scenarios are needed to test the correctness.

- *Benign Datarace* can occur when:
  - *One shared variable-two threads:* 2 threads T1 and T2 share a variable x with 2 different modes Read(R) or Write(W).

```
T1:                              T2:
  A1 x = x +1;                     B1 x = x +1;
  A2 printf("x=%d\n",x);           B2 printf("x=%d\n",x);
```

    Datarace is detected in T2 at line B1 and in T1 at line B1 for the execution sequence A1, B1, A2, B2 and B1, A1, A2, B2 respectively.
  - *Two shared variables-two threads:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
  A1 x = y +1;                     B1 y = x +1;
  A2 printf("x=%d\n",x);           B2 printf("y=%d\n",y);
```

Datarace is detected in T2 at line B1 and in T1 at line B1 for the execution sequence A1, B1, A2, B2 and B1, A1, A2, B2 respectively.

– *Datarace in only one thread:* T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
  A1 pthread_mutex_lock            B1 pthread_mutex_lock
       (&mutex);                        (&mutex);
  A2 x = y +1;                     B2 y = x +1;
  A3 pthread_mutex_unlock          B3 pthread_mutex_unlock
       (&mutex);                        (&mutex);
  A4 x++;
```

Datarace is detected in T1 at line A4 for the execution sequence A1, A2, A3, B1, B2, A4, B3.

– *Datarace in two threads:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
  A1 x = y +1;                     B1 y = x +1;
  A2 y = x;                        B2 x = y;
  A3 printf("x=%d\n",x);          B3 printf("y=%d\n",y);
```

Datarace is detected in T2 at line B1 and in T1 at line A2 for the execution sequence A1, B1, A2, A3, B2, B3. Another is detected in T1 at line A1 and in T2 at line B3 for the execution sequence B1, B2, A1, A2, B3, A3.

- *Fatal Datarace* can occur when:
  – *Case 1:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                                    T2:
  A1 x=(int*)                            B1 x[1]=1;
       malloc(2*sizeof(int));            B2 free(x);
  A2 x[0] = 1;
```

Datarace is detected in T1 at line A2 for the execution sequence A1, B1, A2, B2. Also segmentation fault at A2.

– *Case 2:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
  A1 x++;                          B1 x++;
  A2 if(x == 0){                   B2 if(x == 0){
  A3 free(obj);                    B3 free(obj);
  A4 }                             B4 }
  A5 x--;                          B5 x--;
```

Datarace is detected in T2 at line B1 and in T1 at line A2 for the execution sequence A1:1, B1, A1:2, A2, A3, B2, B3, B4, A4, A5, B5 and Crash will happen at B3. Here A1:1 is "compute x+1" and A1:2 is "assign to x".

– *Case 3:* 2 threads T1 and T2 share a variable x with in Write (W) mode.

```
T1:                              T2:
  A1 x = x+4;                      B1 x=3;
```

Datarace is detected in T2 at line B1 for the execution sequence A1:1, B1, A1:2 where A1:1 is "compute x+1" and A1:2 is "assign to x".

- *Benign/Fatal Datarace* can occur when:
  – *Case 1:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
   A1 if ( x<y ) {                  B1 x=y-1;
   A2 z = x+4;
   A3 y = z;
   A4 }
   A5 z++;
   A6 printf("x=%d",x);
```

Datarace is not detected even though there is potential datarace in the program for the execution sequence B1, A1 , A2, A3, A4, A5, A6.

– *Case 2:* 2 threads T1 and T2 share 2 variables x and y with 2 different modes Read (R) or Write (W).

```
T1:                              T2:
   A1 if ( y<0 ) {                  B1 x=y-1;
   A2 sleep(10);
   A3 y = x+4;
   A4 }
   A5 y++;
   A6 printf("x=%d",x);
```

Datarace detected due to execution of sleep(10) by T1 as there is potential datarace in the program for the execution sequence A1, A2, B1, A3, A4, A5, A6.

– *Case 3:* 2 threads T1 and T2 share variable x with 2 different modes Read (R) or Write (W).

```
T1:                                    T2:
   A1 x=(int*)                            B1 x[1]=1;
        malloc(2*sizeof(int));            B2 sleep(10);
   A2 x[0] = 1;                           B3 free(x);
```

There exists a Fatal Datarace but it is not detected due to execution of sleep (10) by T2 for the execution sequence B1, B2, A1, A2, B3.

6.2 Correctness for Deadlock

Since PGDB works on dynamic analysis, there are cases of deadlock which get overlooked:

- *Deadlock* detected for the following case:
  T1 and T2 share 2 variables x, y with 2 different modes Read(R) or Write(W) with locking variables mutex1, mutex2.

```
T1:
  A1 pthread_mutex_lock(&mutex1);
  A2 pthread_mutex_lock(&mutex2);
  A3 x = y+1;
  A4 pthread_mutex_unlock(&mutex2);
  A5 pthread_mutex_unlock(&mutex1);
T2:
  B1 pthread_mutex_lock(&mutex2);
  B2 pthread_mutex_lock(&mutex1);
  B3 y = x+1;
  B4 pthread_mutex_unlock(&mutex1);
  B5 pthread_mutex_unlock(&mutex2);
```

Deadlock is detected in T2 at line B2 and datarace is also detected in T1 at line A2 for the execution sequence A1, B1, A2, B2, B3, B4, A3, A4, A5, A5 and for sequence A1, B1, B2, A2, A3, A4, A5, B3, B4, B5 respectively. Though potential deadlock exists in the application, no deadlock occurred and hence not detected by PGDB for execution sequence A1, A2, B1, A3, A4, A5, B2, B3, B4, B5.

6.3 Benchmark Testing

To test the behaviour, accuracy and efficiency of PGDB, we have used (with modification) a set of benchmarks of popular dataraces detected by Google's Thread-Sanitizer Tool[5].

- **Case 1:** *No Datarace condition with one thread.* Benchmark has only one thread and does not have datarace. The result will be negative.
- **Case 2:** *No Datarace condition with synchronization.* Benchmark has two threads accessing one global shared variable which is synchronized with proper locking mechanism to prevent datarace. Thus the result will be negative.
- **Case 3:** *Datarace condition with synchronized and non-synchronized shared variables.* Benchmark has two threads with two global shared variables. One global is accessed using locks while the other is accessed without locks. So access to one of them will lead to datarace. One datarace is reported.
- **Case 4:** *Datarace leading to crash due to write to freed memory or double freeing.* Benchmark has two threads accessing a dynamically allocated memory location which is freed based on a reference count and a non-synchronized access to this reference count will lead to being freed more than once hence leading to program crash. Reported in real time applications like Chrome, SQLite etc.
- **Case 5:** *Datarace on Boolean flag used for thread synchronization.* Benchmark has two threads and the synchronization between these two threads is done using a boolean variable which is shared between the threads and due to out of order execution in latest architectures leads to unexpected results. Datarace will be reported on the shared Boolean flag.
- **Case 6:** *No Datarace condition by adding memory barriers for in-order instruction execution.* This benchmark consists of two threads with two shared variables using memory barriers which enforces ordering of memory access for shared data synchronization to solve unexpected results caused due to out of order instruction execution. Result of PGDB should be negative.
- **Case 7:** *Datarace condition due to improper usage of memory barrier instruction.* This benchmark consists of two threads with two shared variables. The access to these shared data is not synchronized due to usage of memory barrier instructions at improper places leading to datarace. Reported by PGDB.
- **Case 8:** *Datarace condition due to Initializing objects without synchronization.* Benchmark has two threads trying to initialize an object by dynamically allocating memory. Since the allocation is done without any synchronization, it might lead to memory leaks. Should be detected by our PGDB.
- **Case 9:** *Datarace on free.* This benchmark has two threads where one thread dynamically allocates memory in the heap whereas the other thread frees this

[5]https://code.google.com/p/thread-sanitizer/wiki/PopularDataRaces

Table 2: Different Scenarios of Barrier

#	Thread 1	Thread 2	Thread 3	Potential of Datarace between
1	`void* func1() { x = 1; pthread_barrier_wait(&bar);}`	`void* func2() { x++; pthread_barrier_wait(&bar); }`	NA	(T1 & T2)
2	`void* func1() { x = 1; pthread_barrier_wait(&bar);}`	`void* func2() { x++; pthread_barrier_wait(&bar);}`	`void* func3() { x = 2; }`	(T1 & T2) or (T2 & T3) or (T1 & T3)
3	`void* func1() { x = 1; pthread_barrier_wait(&bar);}`	`void* func2() { pthread_barrier_wait(&bar); y=x;}`	NA	None
4	`void* func1() { pthread_barrier_wait(&bar); x = 1; }`	`void* func2() { y = x; pthread_barrier_wait(&bar);}`	NA	None
5	`void* func1() { x = 1; pthread_barrier_wait(&bar);}`	`void* func2() { pthread_barrier_wait(&bar); y = x; }`	`void* func3() { x = 2; }`	(T1 & T3) or (T2 & T3)
6	`void* func1() { x = 1; pthread_barrier_wait(&bar); }`	`void* func2() { y = x; pthread_barrier_wait(&bar); }`	NA	(T1 & T2)
7	`void* function1() { pthread_barrier_wait(&bar); x = 1; }`	`void* func2() { pthread_barrier_wait(&bar); y = x; }`	`void* func3() { x = 2; }`	(T1 & T2) or (T2 & T3) or (T1 & T3)

area causing a crash due to datarace. So, the result of PGDB should be positive with reporting of datarace.

- **Case 10:** *Datarace on exit.* This benchmark has two threads created by the main program which are accessing a shared global object where before both the threads end, the main program exits thereby making the shared object unavailable to both of them. Should be detected and reported by PGDB.
- **Case 11:** *Datarace on mutex.* consists of two threads that are synchronized by locking or unlocking a shared mutex and any change in mutex value or its destruction by one thread will affect the other thread still in execution leading to datarace of the mutex. This datarace should be reported by PGDB.
- **Case 12:** *Datarace on file descriptor.* consists of two threads that are accessing the same file descriptor for read/write purpose without any synchronisation leading to data being written on a wrong file or socket causing leaking of sensitive data into an untrusted network connection in real time. Reported by PGDB.

Table 3: PGDB Output for Datarace

Benchmark Case	Expected Output	PGDB Output
Case 1	No Datarace	×
Case 2	No Datarace	×
Case 3	Datarace Exists	✓
Case 4	Datarace Exists	✓
Case 5	Datarace Exists	✓
Case 6	Datarace Exists	✓
Case 7	Datarace Exists	✓
Case 8	Datarace Exists	✓
Case 9	Datarace Exists	✓
Case 10	No Datarace	×
Case 11	Datarace Exists	✓
Case 12	Datarace Exists	✓

✓- Data Race detected × -Data Race not detected

6.4 Performance Testing

The user can selectively turn on or off the instrumentation to increase performance. While instrumentation adds considerable overhead on the execution time, we find that with PGDB's selective instrumentation it can be significantly reduced.

7. Conclusion and Future Work

We have presented strategies to dynamically instrument multi-threaded programs (written in C/C++ using pthreads library) using PIN and to integrate the same with GDB (on Linux) to debug for dataraces and deadlocks, if any. The support has been implemented with new commands in PGDB (Table 1). Going forward we would like to support livelock detection in PGDB, extend the augmentations for Microsoft Visual Studio Debugger [6] on Windows, and support other thread libraries / models like Windows threads [11], Intel TBB [9] and Boost [10]. We would also like to improve the performance of the pintools to make PGDB more effective.

References

[1] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. *Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation* in PLDI '05 Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (Chicago, IL, USA, June 12 - 15, 2005). ACM SIGPLAN Notices, Volume 40 Issue 6, June 2005, Pages 190-200.

[2] Moshe (Maury) Bach, Mark Charney, Robert Cohn, Elena Demikhovsky, Tevi Devor, Kim Hazelwood, Aamer Jaleel, Chi-Keung Luk, Gail Lyons, Harish Patil, and Ady Tal. *Analyzing Parallel Programs with PIN*, Journal Computer, Volume 43, Issue 3, March 2010, Pages 34-41.

[3] PIN User Manual: http://www.pintool.org

[4] GDB: The GNU Project Debugger: http://www.gnu.org/software/gdb

[5] IDB: Intel Debugger: http://software.intel.com/en-us/articles/idb-linux

[6] Debugging in Visual Studio: http://msdn.microsoft.com/en-us/library/vstudio/sc65sadd.aspx

[7] Xiaoming Shi, Venkatesh Karthik Srinivasan, Madhu Ramanathan, and Yiqing Yang. *PinDB: A PIN-based Debugger for Multi-threaded Programming*, http://pages.cs.wisc.edu/ madhurm/pindb/pindb.pdf.

[8] pthreads (POSIX Threads): https://computing.llnl.gov/tutorials/pthreads/

[9] Intel TBB (Thread Building Blocks): http://threadingbuildingblocks.org/

[10] Boost Threads: http://www.boost.org/

[11] Windows Threads: http://msdn.microsoft.com/en-us/library/windows/desktop/ms684847(v=vs.85).aspx

SESSION

SOFTWARE ENGINEERING AND DEVELOPMENT ISSUES + SOFTWARE SCALABILITY + APPLICATIONS

Chair(s)

TBA

Performance of Lambda Expressions in Java 8

A. Ward[1], **and D. Deugo**[1]
[1]School of Computer Science, Carleton University, Ottawa, Ontario, Canada

Abstract – *A major feature introduced to developers in Java 8 is language-level support for lambda expressions. Oracle claims that the use of lambda expressions will allow for the writing of more concise and efficient code for processing elements that are stored in collections. We consider the problem of determining if the runtime of lambda expressions in Java 8 are faster than non-lambda expressions. By comparing the speed of a lambda expression with a corresponding non-lambda expression, that renders the same result, we describe where lambda expressions have performance advantages or disadvantages.*

Keywords: lambda expression, stream, Java 8

1. Introduction

Lambda expressions in computer programming are byproducts of the mathematical logic system, lambda calculus. It was Alonzo Church who came up with lambda calculus [16] to give structure to the concept of effective computability. Lambda expressions are also known *anonymous functions* because in lambda calculus all functions are anonymous (not bound to an identifier). Lambda expressions have now been used in computer programming since their introduction in Lisp in 1958. Forty-six years later lambda expressions are getting introduced to Java programmers.

1.1 Problem

The question we are trying to answer is the following: are lambda expressions in Java 8 faster than non-lambda expressions at accomplishing the same tasks.

1.2 Motivation

Our motivation for this work came from Oracle's claim that the use of lambda expressions would result in more efficient code [1]. We were interested in determining if there were advantages to using lambda expressions beyond their obvious conciseness. Considering Oracle claimed lambda expressions were more efficient, we decided to validate by getting a quantitative speed difference in milliseconds and as a percent.

1.3 Goals

Our main goal is to determine if the newly introduced lambda expressions have a speed advantage over non-lambda expressions for an identical task. We also wanted to make a website that ran our comparisons. The website is also intended to allow users to educate themselves on the performance and uses of lambda expressions.

1.4 Objectives

To meet our goals, we have the following objectives:

- Find example lambda expressions to use for comparisons. These will be found through Java 8 books and Oracle documentation.
- In addition to finding lambda expressions, create our own lambda expressions. The created lambda expressions are to show additional uses for lambda expressions that are not covered in the Java 8 books or in Oracle's documentation.
- Using Eclipse, calculate how long a lambda expression takes to finish its task. Then test the speed of a non-lambda expression completing the same task. These results are used to determine the difference, between the lambda and non-lambda expression, in milliseconds and as a percentage.
- Complete a website that runs and outputs the comparison results. This would allow speed comparisons to be tested across multiple operating systems and processor speeds.
- Make the website user friendly and provide thorough instructions on how to allow the website to run. The purpose of this is to allow users of the site to run their own performance comparisons and see the speed difference for themselves.
- Display the code used for comparisons on the website. This is meant to allow users to learn some ways that lambda expressions can be used. Additionally, giving users the ability to see the code used will allow further discussion about if there is a better, faster way to code the lambda or non-lambda expressions.

1.5 Outline

In section 2 we give a brief background on lambda expressions. In section 3 we give some examples of lambda expressions and compare them against their equivalent non-lambda expression. In section 4, report on performance comparison of the examples discussed in section 3. Finally, in section 5, we give our conclusions.

2. Background

The use of lambda expressions has long been ubiquitous in various functional programming languages such as Lisp,

Scala, Haskell, and F# [2], among others. Before Java 8, Java programmers have been forced into writing more verbose code than needed and lacked key functionality such as the ability to pass in a function as a parameter.

The following subsections give an overview of lambda expressions. This discussion includes what a lambda expression is, why they are useful, and finally their performance in other languages.

2.1 What is a Lambda Expression?

Lambda expressions are also known as *anonymous functions* because they are functions without an identifier. These expressions can make use of already programmed functional interfaces, such as a Predicate or Function. With no identifier, a lambda expression isn't intended to be called many times like a method. They are actually commonly used to avoid coding unnecessary methods. Thus, if the functionality is only needed once or for a short amount of time, lambda expressions help make code clearer and concise [3].

Example:

```
 /*to get the total + tax of a list of prices */
ArrayList<Integer> prices = new
ArrayList<Integer>(Arrays.asList(90,87,34,21));
// using lambda expression
double total = prices.stream()
        .mapToDouble(x -> x*1.14)
        .sum();
//using non-lambda expression
double total2 = 0.0;
for (Integer i : prices) {
    total2 += i*1.14;
}
```

For example, the above lambda expression can be written using one line of code. The non-lambda expression involves first initializing a variable and then creating a for-loop to add each item price, with tax, to the total.

2.2 Why Use Lambda Expressions?

As stated in [3], the use of functional interfaces paired with anonymous inner classes is a common theme in Java. To simplify the coding, functional interfaces are taken advantage of for use with lambda expressions, eliminating the need to program inner classes.

Example:

```
// Using inner class
btn.setOnAction(new EventHandler<ActionEvent>()    {
    @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
  });
// Using lambda expression
btn.setOnAction(event->
System.out.println("Hello World!"));
```

The "*horizontal solution"* of using lambda expressions, solves the "*vertical problem*" presented by using inner classes [3]. A lambda expression addresses the bulkiness of an inner class by converting 5, 6, or even more lines of code into a single statement.

2.3 Performance in Other Programming Languages

Performance of lambda expressions differs from one programming language to another. In some languages, the use of a lambda expression not only gives the code a more clear and concise look, but also has a faster execution time. However, in other programming languages, code runs faster using the more verbose approach [6]. As stated in [6], the "lazy" approach can have its costs when it comes to efficiency; using lambda expressions when needed is slower than calling a function by name. The following subsections review the use of lambda expressions in other popular programming languages.

2.3.1 In Haskell

Haskell is a purely functional programming language, based on lambda calculus. In the release of Haskell version 1.0, in 1990, it was well known that the use of lambda expressions caused a significant and constant performance loss [6]. Haskell 1.0 was also inefficient when it came to defining streams [6], making it more inefficient to use lambda expressions on streams of data.

Haskell Prime was released in 2006, where much of the development focus was on performance [6]. Now competitive performance is available with Haskell [6]. Haskell programmers can now use the functionality of lambda expressions without the inferiority of performance.

2.3.2 In Python

According to Python's official documentation, lambda expressions are equivalent to regular function objects [8]. In Python, lambda expressions are just a better syntactic way to write a normal function [9]. Thus, the implication is that lambda expressions have equivalent performance as non-lambda expressions in Python. However, as stated in [10], lambda expressions in Python can be more efficient to use for common programming idioms such as mapping, filtering, and list comprehension.

2.3.3 In C++

C++11 was released in 2011 and saw major revisions including the use of lambda expressions [11]. According to the ISO (International Organization for Standardization), the addition of lambda expressions to C++ has added much strength flexibility and efficiency [11]. C++ programmers are

now enabled to use powerful expressiveness, and write efficient, high performance code [11].

3. Lambda Comparisons

Lambda expressions allow for a much more concise way of iterating over a collection of data such as a list. Lambda expressions can use multiple functions and interfaces to accomplish a task. The following subsections go over some of the varying ways that lambda expressions can be used.

3.1 Reduction

The stream.reduce() method is a general reduction operation. It is comprised of an *identity* element that is the starting value of the reduction and default value if there are no elements in the stream. The method also consists of an *accumulator* that takes as parameters, the partial result so far of the reduction, and the next element in the stream. A new partial result gets returned.

```
Example: int total = nums.stream()
                    .reduce(0, (a, b) -> a+b);
```

The non-lambda way of accomplishing the same thing as in the above example would be creating a variable to store the sum, and then running a for-each loop where each number in the list was added to the total.

```
Example: int total = 0;
         for (int i : nums) {
             total += i;
          }
```

What would have taken 3 lines of code in previous Java versions, can now be done in 1 line using lambda expressions.

3.2 Filtering

The stream.filter() method takes a predicate as an argument and returns a new stream containing the elements that matched the conditions of predicate. Each predicate can have multiple conditions that need to be satisfied. A lambda expression can be passed into the stream.filter() method instead.

```
Example: List<String> filtered = strList.stream()
                    .filter(x -> x.length()> 3)
                    .collect(Collectors.toList());
```

The above example filters out all strings with a length less than 3. This creates a new stream with only the remaining strings. The non-lambda expression once again uses a for-each loop. Instead of filtering, the non-lambda expression uses an if-statement.

```
Example: List<String> filtered = new
                    ArrayList<String>();
         for (String str : strList) {
             if (str.length() > 3)
                 filtered.add(str);
         }
```

Once again the lambda expression can be written using less lines of code.

3.3 Collecting

The stream.Collectors class has a variety of methods that are of great use to streams and lambda expressions. These methods from the Collectors class can be used inside the stream.collect method of the lambda expression.

3.3.1 To List

One method of the Collectors class is the Collector.toList() method. The method takes all the elements that are left in a stream, and stores them in a list. This makes it quick and easy to create a new list, filtering out unwanted elements from the old list. For example, extracting all the numbers in a list that are greater than 5.

```
Example: List<Integer> above5 = numberList
                    .stream()
                    .filter(x -> x > 5)
                    .collect(Collectors.toList());
```

In the above example, with 1 line of code a new list is created, containing only the desired numbers. A non-Lambda expression to accomplish the same feet requires creating a new list, using an if-statement to check the value of each number, and then adding the wanted numbers to the List.

```
Example: List<Integer> above5 = new
                        ArrayList<Integer>();
         for (Integer i : numberList) {
             if (i > 5)
                 above5.add(i);
         }
```

3.3.2 Joining

Another method in the Collectors class is the joining() method. The method is a terminal operation that creates a non-stream result. Inside the stream.collect method, the joining() method returns a Collector that concatenates all the elements in the stream. The joining() method can take a CharSequence as a parameter. In that case a Collector is returned that concatenates the stream elements with the CharSequence separating each element.

```
Example: String con = names.stream()
                        .collect(
                        Collectors.joining(", "));
```

In the above example, in 1 line of code, the joining() method creates a concatenated string with a comma separating

each element. Using a non-lambda expressions involves first creating a blank string and then using a for-loop to add each list element to the string.

```
Example: String con = "";
         for (int i = 0; i < names.size(); i ++) {
             con += names.get(i) + ", ";
         }
```

3.4 Mapping

Mapping involves taking an object and assigning it to a new value. If for example there was a stream filled with Person objects, mapping could create a stream filled with numbers, such as the Person object's age. Mapping can be accomplished with the stream.map() method. The method can take a lambda expression as a parameter.

```
Example: List<String> upperNames =  names
                .stream()
                .map(name -> name.toUpperCase())
                .collect(Collectors.toList());
```

The above example streams a list of strings and maps each string to a string with all uppercase letters. The equivalent non-lambda expression creates a new list, and then iterates through a for-loop of the original list. Each string from the original list is converted to all upper case letters before being added to the new list.

```
Example: ArrayList<String> upperNames = new
                        ArrayList<String>();
        for (String name : names) {
            upperNames.add(name.toUpperCase());
        }
```

3.5 Passing In Functions and Predicates

As previously mentioned, a feature lacking in previous Java versions was the ability to pass in functions. The java.util.function package can be used to pass in a Function or Predicate into a stream's intermediate operation(s) to replace a lambda expression. Both a Function and Predicate can return true or false, allowing them to passed in to methods that need to evaluate a condition (i.e. the stream.filter() method).

3.5.1 Predicates

A predicate is a functional interface that can be used as a target for a lambda expression or method reference. The syntax for defining a predicate is Predicate<T> where T is the type of argument being tested (i.e String, int, etc). The Predicate<T> then determines if the input object meets some criteria.

```
Example: Predicate<String> startWithA = (p) ->
                (p.startsWith("A"));
            List<String> startingWithA =
                names.stream()
                .filter(startWithA)
                .collect(Collectors.toList());
```

In the above example a predicate named 'startWithA' is created. The predicate is then passed in to filter the stream. As mentioned in section 3.2 on filtering, the non-lambda expression equivalent involves an if-statement and a for loop.

```
Example: List<String> startingWithA = new
                            ArrayList<String>();
        for (String name : names) {
            if (name.startsWith("A")) {
                startingWithA.add(name);
            }
        }
```

3.5.2 Functions

A function is also a functional interface that can be used as a target for a lambda expression or method reference. The syntax for defining a function is Function<T, R> where T is the type of argument being passed in and R is the type of result for the function. The Function<T, R> takes in a single argument and returns some result. Unlike the predicate the result isn't necessarily a Boolean.

```
Example: Function<String, Predicate<String>>
                startsWithLetter = letter -> name ->
                name.startsWith(letter);

            List<String> namesStartingWithA =
                            names.stream()
                .filter(startsWithLetter.apply("A"))
                .collect(Collectors.toList());
```

In the above example, a predicate is returned by the function. What makes the function different from the predicate example in section 3.4.1 is the ability to check if the string started with any letter. Whereas the predicate in section 3.4.1 was hard coded to only check if the string started with the letter "A". An equivalent non-lambda expression involves creating a separate method inside the class file.

```
Example: List<String> namesStartingWithA = new
                            ArrayList<String>();
        for (String name : names) {
            if (startsWith("A", name))
                namesStartingWithA.add(name);
        }

    public boolean startsWith(String a, String b) {
        return b.startsWith(a);
    }
```

3.6 Calling Class Methods

Lambda expressions can be used to call methods written elsewhere in the class or superclass. The method could return

a boolean for filtering, be used for mapping, or be part of the terminal operation. For the comparison on the website we used the following method:

```
static boolean isPrime(int n) {
   for(int i=2;i<n;i++) {
      if(n%i==0)
         return false;
   }
   return true;
}
```

Since a Boolean is returned, the isPrime() method is used in the body of a lambda expression. The lambda expression is inside the stream.filter() method to filter the stream of numbers. The stream.count() method is then used to add up how many elements are left in the stream.

Example: *int counter = (int) nums.stream()*
.filter(p -> isPrime(p)).count();

Yet again the non-lambda expression involves the use of a for-loop. An integer is created that keeps track of the total number of prime numbers. The for-loop iterates through each number in a list. The isPrime() method is used inside an if-statement. If the number is prime, 1 is added to the total.

```
Example: for (int n : nums) {
             if (isPrime(n)) {
                 counter++;
             }
         }
```

4. Results

In this section we provide comparisons of the execution times of the examples noted in the previous section.

4.1 Comparison Results

The data presented in the Tables 1 and 2 are the result of executing each lambda expression and non-lambda expression for problems of size 10,000, repeating each experiment 1000 times and then averaging the results using a Mac Pro laptop, 2.9 GHz Intel Core i7, 8 GB 1600 MHz, DDR3 memory, with Java 8. The average execution time in milliseconds and the ratio of improvement between the lambda and non-lambda expressions are noted in Table 1 and 2. The Lambda improvement is calculated as follows: (Lambda - Non-Lambda) / (Non-Lambda) * -100.00. To remove any startup or Just In Time (JIT) effects [17], the results report in Table 1 where from the fifth iteration of running the above experiments. We found that by the third iteration the results were consistent with iterations four and five. Table 2 shows the results of the first iteration, which are considerably different from the results reported in Table 1. Table 3 shows how drastically a small problem size and only one iteration can affect the results. The results in this table show how using a problem size of 1000, only running each experiment 100 times, and then looking at the first iteration of this impacts the performance of Lambdas. To run your own comparisons, visit http://people.scs.carleton.ca/~deugo/java8

Table 1: Lambda Performance Comparisons (5'th Iteration)

Experiment	Lambda (ms)	Non-Lambda (ms)	Lambda Improvement (%)
Counting Primes	16.81	16.42	-2.40
Adding Up Numbers	16.81	16.42	-2.40
Concatenating Strings	32.78	73.31	55.29
Mapping	70.58	105.80	33.29
Filter List	72.91	106.18	31.23
Filter List with Predicate	79.47	107.25	25.90
Filter In List Function	87.13	108.32	19.57

Table 2: Lambda Performance Comparisons (1'st Iteration)

Experiment	Lambda (ms)	Non-Lambda (ms)	Lambda Improvement (%)
Counting Primes	15.96	15.32	2.25
Adding Up Numbers	15.96	16.33	2.25
Concatenating Strings	30.44	72.82	58.20
Mapping	66.9	105.19	36.40
Filter List	69.21	105.54	34.42
Filter List with Predicate	74.71	106.63	29.93
Filter In List Function	81.15	107.71	24.66

Table 3: Lambda Performance Comparisons (small problem size and repetitions)

Experiment	Lambda (ms)	Non-Lambda (ms)	Lambda Improvement (%)
Counting Primes	0.72	0.01	-7100
Adding Up Numbers	0.74	0.01	-7300
Concatenating Strings	2.51	9.34	73.13
Mapping	6.28	12.24	48.70
Filter List	6.47	12.37	47.70
Filter List with Predicate	6.51	12.5	47.92
Filter In List Function	6.69	12.63	47.03

5. Conclusions

The addition of lambda expressions to Java 8 provide for more functional, concise, and readable coding. In addition, given enough execution time, the new lambda expressions can provide a performance advantage. The report entitled 'Clash of the Lambdas' also shows that Java's lambda expressions not only held their own, but in many cases outperformed the lambda expressions in Scala, C#, and F# [15]. These results held true for Windows and Linux, and varying processor speeds. This is impressive on many levels considering lambda expressions have been present in Scala and F# since their introductions, and in C# since C# 3.0. With this being Java's first attempt at lambda expressions, the results are impressive.

5.1 The Future

With Java 8 being the first version of Java with support for lambda expressions, the future looks promising. Looking at the performance of lambdas in other programming languages, Java's performance not only competes, but also leads over other languages. More impressively, some of those languages have supported lambda expressions for years. Java 9 was announced for release in 2016. This provides another opportunity for Oracle to continue to increase the performance advantages of lambda.

6. References

[1] Gallardo, R. JDK 8 Documentation - Developer Preview Release (The Java Tutorials Blog). Oracle Blogs, 9 Sept. 2013. Web 9 May 2015. <https://blogs.oracle.com/thejavatutorials/entry/jdk_8_documentation_developer_preview>.

[2] Odersky, M., & Spoon, L. Programming in Scala (2nd ed.). Walnut Creek, Calif.: Artima, 2010.

[3] Williams, M., & Nunez, J. Q. (n.d.). Java SE 8: Lambda Quick Start. Oracle. Web 9 May 2015. <http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html>.

[4] Subramaniam, V. Functional programming in Java: harnessing the power of Java 8 Lambda expressions. United States of America: The Pragmatic Programmers, 2014.

[5] Lesson: Aggregate Operations. The Java Tutorials. Web 9 May 2015. <http://docs.oracle.com/javase/tutorial/collections/streams/>.

[6] Hudak, P., Hughes, J., Jones, S. P., & Wadler, P. A History of Haskell: Being Lazy With Class. 16 April 2007. Web 9 May 2015. <http://research.microsoft.com/en-us/um/people/simonpj/papers/history-of-haskell/history.pdf>.

[7] Peyton Jones, Simon, ed. Haskell 98 Language and Libraries: The Revised Report. Cambridge University Press. 2003.

[8] 6. Expressions. Python 3.4.1 documentation. Web 9 May 2015. <https://docs.python.org/3/reference/expressions.html>.

[9] 4. More Control Flow Tools. Python 3.4.1 documentation. Web 9 May 2015. <https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions>.

[10] Erdmann, R. G. Map, Filter, Lambda, and List Comprehensions in Python. Web 9 May 2015. <http://www.u.arizona.edu/~erdmann/mse350/topics/list_comprehensions.html>.

[11] Lazarte, M. C++ language gets high marks on performance with new ISO/IEC standard (2011-10-10). ISO News. 10 Oct. 2011. Web 9 May 2015. <http://www.iso.org/iso/home/news_index/news_archive/news.htm?refid=Ref1472>.

[12] Hejlsberg, A., & Torgersen, M. Overview of C# 3.0. Microsoft Developer Network. 1 Apr. 2007. Web 9 May 2015. <http://msdn.microsoft.com/en-us/library/bb308966.aspx>.

[13] Kennedy, A. C# is a functional programming language. Microsoft Research Cambridge. Web 9 May 2015. <http://sneezy.cs.nott.ac.uk/fun/nov-06/FunPm.pdf>.

[14] Parallelism. The Java Tutorials. Web 9 May 2015. <http://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>.

[15] Biboudis, A., Palladinos, N., & Smaragdakis, Y. Clash of the Lambdas. Web 9 May 2015. <http://cgi.di.uoa.gr/~biboudis/clashofthelambdas.pdf>.

[16] A. Church, A set of postulates for the foundation of logic, Annals of Mathematics, Series 2, 33:346–366

[17] The JIT Compiler, Web 9 May 2015. <http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.aix.80.doc/diag/understanding/jit.html>.

Programmars: A Revolution in Computer Language Parsing

Steven A. O'Hara, PhD
Eagle Legacy Modernization, LLC
702 Southwick Avenue
Grovetown, GA 30813
steve@eaglegacy.com

Topical keywords: parser, grammar, software analysis

Abstract - *This paper presents a revolutionary way to parse computer programming languages without a traditional grammar. The motivation behind this approach is to dramatically increase scalability. The intention is to be able to parse and analyze billions of lines of code written in hundreds of programming languages. To achieve that goal, it is advantageous to have sharable, open-source, modular ways for defining the syntax and semantics of programming languages. The new parsing technique replaces a traditional grammar with a computer program, referred to as a Programmar (short for program and grammar). All the basic operations in BNF (sequencing, alternation, optional terms, repeating and grouping) are supported, and the Java code is both sharable and modular. This parsing approach enables dozens or even hundreds of developers to work on computer program analysis concurrently, while avoiding many of the consistency issues encountered when building grammars and associated code analysis tools.*

1 Introduction

Businesses around the world today collectively have billions of lines of production software written in legacy computer languages like COBOL, RPG, PL/I, Fortran and Natural. These organizations are highly motivated to modernize their software for a number of reasons, including difficulties in maintaining old, brittle code [1] and in hiring people with legacy skillsets [2]. Unfortunately the modernization process is often either prohibitively expensive or produces new software of low quality that is difficult to maintain going forward into the future [3]. Available modernization tools (e.g. [4 to 8]) tend not to be scalable enough to handle large, complex software systems that can be comprised of tens of millions of lines of code written in multiple programming languages.

For the past several decades, legacy software analysis tools have been typified by the type of parser generated by Yet-Another-Compiler-Compiler (YACC) [9]. Such a parser interprets computer program code based on a Context-Free Grammar, which is a declarative description of the syntax of a specific programming language. This parsing process relies on a separate token pre-processor (typically LEX, the Lexical Analyzer [10]) and generates an Abstract Syntax Tree (AST).

Modern programming languages also continue to evolve and require solid analysis approaches (e.g. [11 to 13]). For example, managing deprecated code often requires detailed software analysis similar to application modernization. Unfortunately, there are many one-off grammars and tools for source code analysis, but no standard or shared tools that work well across many programming languages at the same time. With our technique, we process languages as disparate as Java, HTML, CSS, DOS, XML, COBOL, Natural and RPG using a single parser.

This paper introduces a new parsing technique that embeds all required parsing information within a Java program. All of the elements needed to describe the computer programming language(s) to be parsed are embedded in the Java program as fields, classes or methods within Java classes. The focus of this work to date is on parsing languages in the context of legacy application modernization.

Grammar rules can be separated into two categories, those that depend on other rules (defining non-terminals) and those that consume characters in the input stream (terminals). Examples of terminals are string literals, comments, numbers, keywords, and punctuation.

In the Programmar approach, Java methods are used for parsing terminals, while classes are defined to enable parsing of non-terminals. The Programmar API uses Java reflection [14] to dynamically infer a grammar while parsing.

2 Comparison to Old Grammars

This section describes the relationship between a traditional BNF-like grammar and our new Programmar.

Sequences

Given a BNF production rule of the form

```
<A> ::= <X> <Y> <Z>;
```

This is represented in a Programmar as in Figure 1.

```
public class A extends TokenSequence {
     public X x;
     public Y y;
     public Z z;
}
```

Figure 1: Sequencing Programmar example

Note that the elements are anonymous in the BNF grammar. The BNF production rule indicates only that an instance of type A can consist of an (unnamed) instance of type X, followed by an instance of type Y, and then an instance of type Z. In Figure 1, however, the instances are named, which means these specific instances can be referred to from elsewhere in the Programmar or in associated source code analysis programs. This turns out to be extremely valuable when analyzing computer source code.

Alternation

Given a BNF production rule of the form

```
<A> ::= <X> | <Y> | <Z>;
```

This is represented in a Programmar as in Figure 2.

```
public class A extends TokenChooser {
     public X x;
     public Y y;
     public Z z;
}
```

Figure 2: Alternation Programmar example

This means that exactly one of the three elements must be present to be recognized as an instance of type A. As a convenience, anonymous inner classes can be used as well.

Optional Terms

Given a BNF production rule of the form

```
<A> ::= <X> [<Y>] <Z>;
```

This is represented in a Programmar as in Figure 3.

```
public class A extends TokenSequence {
     public X x;
     public @OPT Y y;
     public Z z;
}
```

Figure 3: Optional item Programmar example

Repeating Terms

Given a BNF production rule of the form

```
<A> ::= <X> <Y>* <Z>;
```

This is represented in a Programmar as in Figure 4.

```
public class A extends TokenSequence {
     public X x;
     public @OPT TokenList<Y> y;
     public Z z;
}
```

Figure 4: Repeating term Programmar example

The '+' BNF notation is handled in a similar manner, without the @OPT.

Terminal Nodes

Java code is provided by the Programmar API to assist parsing the most common terminal nodes. For example, string literals in various programming languages commonly have a number of features such as:

- Single or double quotes?
- Are pairs of quotes treated as single quote?
- What is the escape character, if any?
- Can a literal span multiple lines?

Similar routines are available for comments, numbers, punctuation, etc. By using Java code for the terminal nodes, the parsing speed is greatly improved. In our experience, writing a BNF-like grammar for a floating point number or a string literal can be challenging and time consuming.

It is not necessary to use one of the built-in methods for parsing terminal nodes. For example, Python has very strict rules for indentation. Rather than pre-processing the input stream, a Start-of-line terminal node can be used to handle the indentation logic correctly.

3 Motivating Example – Old Grammar

Consider the two PERFORM statements in Figure 5.

```
000160 READ-SHARED-LOCK.
000170     READ SHARED WITH LOCK.
000180     IF WS-STATUS = "00"
000190         GO TO READ-SHARED-EXIT.
000200     IF WS-STAT1 = "2" OR "3"
000210         MOVE 33 TO WS-F-ERROR
000220         PERFORM READ-ERROR.
000230     IF RECORD-LOCKED
000240         PERFORM LOCK-USERS-REC
000250             THRU LOCK-REC-EXIT
000260             WS-COUNT TIMES
000270         ADD 1 TO WS-COUNT
000280         IF WS-COUNT > 25
000290             MOVE 1 TO WS-COUNT
000300         END-IF.
```

Figure 5: Sample COBOL code

In a traditional grammar, the PERFORM verb in COBOL might be expressed as in Figure 6 (this is greatly simplified).

```
cPerform ::= "PERFORM" cParagraph
      [("THROUGH" | "THRU") cParagraph]
      [cTimes];
cTimes ::= cExpression "TIMES";
cParagraph ::= cIdentifier;
cExpression ::= cIdentifier | cNumber;
cIdentifier ::= cLetter
      (cLetter | cDigit | "-")*;
cNumber ::= cDigit+;
cLetter ::= "A" .. "Z";
cDigit ::= "0" .. "9";
```

Figure 6: Traditional grammar example

Once the COBOL program has been parsed, tools can be used to analyze the AST. Typically, such tools traverse the tree looking for specific named entities, such as cPerform. These tools depend heavily on the names used in the grammar. If somebody changes the name of an element in the grammar, there is no easy way to detect that change.

Grammars used during a modernization effort tend to require significant changes when another effort begins, for a variety

of reasons. This can happen because of language variations, hardware variations, operating system variations, business-specific conventions, etc. The tools depend heavily on the terms in the grammar, and the terms are in flux, so it is very easy for grammars to get out of sync with the analysis tools.

This is the main reason why parsers and grammars do not scale well. Minor changes to grammars can have many subtle adverse effects on how parsers work, as well as the tools that perform subsequent processing tasks.

Generally, traditional grammars and parsers are successful when there are only a few people working on them, and they are working on only one or two computer languages. Most large-scale businesses deal with dozens of computer languages. Getting a single parser to handle all languages is difficult, because many languages require significant pre-processing. Also, getting some level of consistency between AST's is difficult.

Our primary goal of parsing all major computer languages in a unified manner is only possible when the grammar and the analysis tools are written in the same language. This is the basis for our claim of scalability with Programmars.

4 Motivating Example – Programmar

With a Programmar, the rules related to COBOL PERFORM verbs could be expressed in a language like Java as in Figure 7. Again, this is a simplified version that does not reflect all possible PERFORM variations.

```
class CobolPerform extends CobolStatement {
  CobolKeyword PERFORM =
      new CobolKeyword("PERFORM");
  CobolParagraph startPara;
  @OPT CobolPerfThrough through;
  @OPT CobolPerfTimes times;

  class CobolPerfThrough extends TokenSequence {
    CobolKeywordList THRU =
        new CobolKeywordList("THRU", "THROUGH");
    CobolParagraph endPara;
  }

  class CobolPerfTimes extends TokenSequence {
    CobolExpression count;
    CobolKeyword TIMES =
        new CobolKeyword("TIMES");
  }
}
```

Figure 7: Simplified Programmar example

Terms like TokenSequence are built in to the Programmar API, and terms like CobolKeyword, CobolExpression, etc. are defined in other Java classes.

The Java program representation shown serves two distinct purposes. First, the program can be considered a grammar for defining the PERFORM statement in COBOL, describing all the different ways the statement can be formed. Second, rather than creating an AST, the parsing process creates instances of this type of class. Collectively these instances form a Programmar Semantic Tree (PST). The PST can be stored as an XML file or as Java code that regenerates it.

5 Advantages

Although the Programmar approach is slightly more verbose than using a context-free grammar (CFG), the new approach offers four major advantages:

Advantage #1 – Dependencies are caught automatically

In the Programmar approach, downstream impact of a change to any part of a Progammar will be detected immediately, because the Java Programmar won't compile. If somebody were to change the name of an element in the Programmar, all references to that name would become invalid until they were updated to be consistent with the changed element. This allows projects to scale to much more significant levels. It is now possible to have dozens of developers working on the same project, processing many computer languages.

Advantage #2 – Semantics: Entities are not just names

In the AST version, a cParagraph is just an identifier. There is no further information attached to it. If you write a tool to analyze or transform a COBOL program, you will have to search the rest of the AST to find out what is in that other paragraph. With the PST version, the CobolParagraph instance contains within it a reference to the actual definition of that paragraph, including all of its statements, line numbers, references, etc. This greatly simplifies the task of writing analysis and conversion tools. Some of the work in connecting references to definitions is accomplished as part of the parsing process, which lessens the effort required to create tools for subsequent processing tasks.

Advantage #3 – Context-Sensitive, not Context-Free

The terminal nodes in a Programmar, such as CobolKeyword, are represented in Java code. When trying to parse a terminal node, the Java code has access to the current context. In Figure 8, COBOL level numbers are very important. A level number 10 following a level 05 means that the 10 should be stored in a sub-tree under the 05. But the next 10 should be stored under the same 05. In a Programmar, COBOL level number logic can examine the context to decide how to correctly build the hierarchy, without any post-processing.

```
01  INV-L7.
  05  FILLER          PIC  X(02).
  05  INV-REMKS.
    10  INV-SM        PIC  X(09).
    10  INV-SMAN      PIC  X(31).
  05  FILLER          PIC  X(05).
  05  INV-RMKS.
    10  FILLER        PIC  X(17).
    10  INV-H18       PIC  X(12).
    10  INV-TERMS.
      15  INV-ADV     PIC  Z(06)9.99-.
```

Figure 8: Traditional grammar example

Advantage #4 – Utilize modern programming languages

Programmars are expressed in a modern programming language such as Java, which means the development methodologies associated with such a modern language can be used when working with Programmars. We identify the five following benefits.

Abstraction. A traditional grammar is typically built to describe just one programming language. With the Programmar approach, the components common to all variations of a particular programming language can be placed into an Abstract Programmar class. For example, there are major variations of languages like Report Program Generator (RPG). A File specification has the same meaning across each variation, so an abstract RPGFile class can be used to define the common elements. The minor syntactic differences between RPG variations can then be represented by concrete classes that extend the abstract class.

Inheritance. Frequently, there are variations on a computer programming language. For example, there are both fixed width (80 column) and free-format COBOL programs. Their meanings are virtually identical, but the syntax is very different. With a traditional grammar, the whole grammar might get copied and edited for each variation. With Programmar Inheritance, only the local changes need to be considered and the rest can be inherited from the main Programmar.

Encapsulation. Some computer languages, such as HTML for web pages, often include other languages inside of them; Javascript, CSS or PHP in the case of HTML. In a traditional grammar, these are normally combined into a monolithic grammar covering all sub-languages. With Programmar Encapsulation, the main Programmar (e.g., HTML) can simply reference the other Programmar (e.g., Javascript).

Logic. With Programmar logic, the full power of the programming language used to represent the Programmar (e.g., Java) is available for complicated cases. Managing the PICTURE level numbers in COBOL is a good example where logic is needed to assist the parsing process to build the correct hierarchy.

Shared Processing. Most terminal tokens are somewhat similar across programming languages. Generic processors for numbers, literal strings, punctuation, etc. are all made available by the Programmar API to use when writing Programmars. For example, parsing functionality for hexadecimal (hex, base 16) numbers can generally be implemented in just a few lines of Java by extending the generic hex number processor, and simply declaring their hex prefix or suffix. Comments, floating point numbers and string literals are simpler to implement in a Programmar than a traditional grammar, because they can utilize the built-in generic methods.

6 Programmar Token Types

Every element in a Programmar is an AbstractToken in the representation of a programming language. The following are the main types of Abstract Tokens.

TokenSequence: Sequence of Tokens

When a Programmar class extends TokenSequence, an instance of this type is identified during parsing when all of the fields in the class are present in the order specified (unless they are marked as optional). Inner classes are a convenient way to define sub-rules for such an element.

TokenChooser: Choose One Token

Programmar classes that extend TokenChooser have both fields and inner classes, and the parser will attempt to match both. Each can be marked with @FIRST or @LAST because the parser makes three passes. The first looks only at @FIRST elements, the second looks at neither @FIRST or @LAST elements, and the third pass looks only at @LAST elements. This gives another level of control over the parser, and can be used to speed up the parser.

Care must be used in the order of the elements in a TokenChooser. Once a token matches from the list of choices, no other tokens are considered. This is done for efficiency purposes and doesn't seem to impose any restrictions other than being careful with ordering. BNF-like rules such as "<A> ::= <X> | <X> <Y>;" need to be written in the other order.

TokenList: One or More Tokens

A field may be a TokenList of another token type, typically a TokenSequence. It will match one or more instances of that type (or zero if @OPT is also present).

PrecedenceToken: Operator Precedence

This is a specialized token to help represent expressions in most computer languages. It allows a declarative specification of unary operators (like minus and not) and binary operators (like plus and times). The order of the elements in the specification determines their precedence. In most BNF-like grammars, this is a complicated process and often verbose. The Programmar approach includes a short-cut way to define operator precedence, which reduces complexity.

The parser also handles the "left-recursion" problem. A Programmar class can represent a BNF-like rule (essentially) as "expr ::= expr + expr" without getting into an infinite loop.

Typical AST's built from parsing expressions can be very long. Our parsing process eliminates needless intermediate layers, which greatly reduces the size and complexity of the resulting PST. For example, if there is no multiplication in the expression, then there is no multiplication node in the PST.

UnparsedElement: Allow Parse to Continue

In some situations, we may know there will be some statements we might not be able to parse. In that case, we can

add an @LAST UnparsedElement to a TokenChooser. This may allow parsing to continue in the event of a parse failure, resulting in a "soft" parse failure.

Optional Element

@OPT is used to indicate that a Token is optional. It is often used in conjunction with a TokenList to mean zero or more elements instead of one or more.

Terminal Tokens

There are several types of pre-defined terminal tokens, such as:

- TerminalLiteralToken
- TerminalNumberToken
- TerminalPunctuationToken
- TerminalKeywordToken
- TerminalCommentToken

Each of these provides built-in generic parsers to support most programming languages. Furthermore, each has an associated CSS style for the code inspection modules so it is easy to visually identify all the literals, numbers, comments, etc. in a computer program listing.

7 How Programmars Work

The central idea behind parsing with Programmars is to use Reflection to fill in the PST with the results of the parsing process. It uses a top-down approach with no look-ahead (i.e., it is an LL(0) grammar [15]). No token pre-processor is required.

The use of a Programmar differs greatly from a Recursive Descent Parser (RDP) [15] because Programmars use a declarative way of representing computer languages and rely on Java reflection to decide how to parse. Other than terminal nodes, there is no logic in a Programmar. A RDP, in contrast, uses programming logic for matching each and every node in the grammar.

The Programmar parsing process is context-sensitive for terminal nodes in the sense that the current (partial) parse tree is accessible to the terminal node parser. For example, PL/I level numbers are hierarchical like COBOL level numbers and they are parsed by looking at what is already in the parse tree.

Reflection is heavily used in a Programmar parser. The parser examines all the data fields and classes inside a given class. Depending on the type of Abstract Token, a different strategy is used. For a Token Sequence, all the elements must be present in the given order. If any one element doesn't match, the whole Token Sequence fails. Recursion is also heavily used since Abstract Tokens may contain other Abstract Tokens.

There is no grammar (other than the Java program), and there is no AST. The result is a programmatic representation of the original source program. This technique has been demonstrated to parse dozens of computer programming languages such as Assembler, Fortran, PL/I, RPG, Java, Visual Basic, Delphi, DOS, SQL, Python, C++, and many more.

One of the available outputs of the Programmar parser is a traditional grammar. In other words, given the Programmar representation of the COBOL programming language, the system can automatically generate a traditional BNF-like grammar from it.

8 JSON Example

Javascript Object Notation (JSON) is a simple language, so it is convenient to use for an abbreviated example, with some key parts omitted from the Programmar shown in Figure 9.

```
public class JSON_Program extends Language {
  public TokenList<JSON_Element> elements;
}

public class JSON_Element extends TokenChooser {
  public JSON_Literal literal;
  public JSON_Number number;
  public JSON_Object object;
  public JSON_Dict dictionary;
  public JSON_KeywordChoice constants = new
    JSON_KeywordChoice("null", "true", "false");
}

public class JSON_Object extends TokenSequence {
  public JSON_Punctuation leftBracket = new
    JSON_Punctuation('[');
  public @OPT JSON_Element element;
  public @OPT TokenList<JSON_MoreElements> more;
  public JSON_Punctuation rightBracket = new
    JSON_Punctuation(']');
}

public class JSON_Dict extends TokenSequence {
  public JSON_Punctuation leftBrace = new
    JSON_Punctuation('{');
  public @OPT JSON_Entry entry;
  public @OPT TokenList<JSON_MoreEntries> more;
  public JSON_Punctuation rightBrace = new
    JSON_Punctuation('}');
}

public class JSON_Entry extends TokenSequence {
  public JSON_Literal name;
  public JSON_Punctuation colon = new
    JSON_Punctuation(':');
  public JSON_Element value;
}
```

Figure 9: JSON Programmar

```
[
 {
   "pk": "1",
   "model": "fixtures_regress.absolute",
   "fields": {
     "name": "Load Absolute Path Test"
   }
 }
]
```

Figure 10: Sample JSON source code

Consider the sample JSON source code in Figure 10. The trace in Figure 11 shows the parsing process. "Next" is the current character sequence. "Pattern" is the name of the Programmar class. The leading periods show the parsing recursion depth.

```
Next    Pattern
======  ======================================
[       ? JSON_Program
[       .? JSON_Element
[       ..? JSON_Literal ()
[       .. Failed JSON_Literal ()
[       ..? JSON_Number
[       .. Failed JSON_Number
[       ..? JSON_Object
[       ...? JSON_Punctuation ([)
{       ... ***** Match JSON_Punctuation ([)
{       ...? JSON_Element
{       ....? JSON_Literal ()
{       .... Failed JSON_Literal ()
{       ....? JSON_Number
{       .... Failed JSON_Number
{       ....? JSON_Object
{       .....? JSON_Punctuation ([)
{       ..... Failed JSON_Punctuation ([)
{       .... Failed JSON_Object
{       ....? JSON_Dict
{       .....? JSON_Punctuation ({)
"pk":   ..... ***** Match JSON_Punctuation ({)
"pk":   .....? JSON_Entry
"pk":   ......? JSON_Literal ()
: "1"   ..... ***** Match JSON_Literal ("pk")
: "1"   ......? JSON_Punctuation (:)
"1",    ..... ***** Match JSON_Punctuation (:)
"1",    ......? JSON_Element
"1",    .......? JSON_Literal ()
,       ..... ***** Match JSON_Literal ("1")
,       ..... ***** Match JSON_Element
,       ..... ***** Match JSON_Entry
    (59 lines omitted)
]       ..... ***** Match JSON_Punctuation ({)
]       .... ***** Match JSON_Dict
]       ... ***** Match JSON_Element
]       ...? JSON_MoreElements
]       ....? JSON_Punctuation (,)
]       .... Failed JSON_Punctuation (,)
]       ... Failed JSON_MoreElements
]       ...? JSON_Punctuation (])
(EOF)   ... ***** Match JSON_Punctuation (])
(EOF)   .. ***** Match JSON_Object
(EOF)   . ***** Match JSON_Element
(EOF)   . ? JSON_Element
(EOF)   ***** Match JSON_Program
```

Figure 11: Trace of the parsing process

To parse a JSON_Program, the parser first tries to match a JSON_Element, which has to be a JSON_Literal, or a JSON_Number, etc. The parser eventually matches on a JSON_Object, at the fourth line from the bottom in Figure 11.

Each "?" in Figure 11 represents one parsing step. Each step should be considered a parsing attempt, which may or may not match the input text stream.

Figure 12 is an abbreviated version of the generated XML version of the PST. Starting and ending character and line positions are in the actual XML file for every token.

```
<Program Elapsed="1" Steps="52" Tokens="35">
 <Token TT="JSON_Program">
  <Token Name="elements" TT="List">
   <Token TT="JSON_Element">
    <Token TT="JSON_Object">
     <Token Name="leftBracket"
       TT="JSON_Punctuation" V="["/>
     <Token Name="element" TT="JSON_Element">
      <Token TT="JSON_Dict">
       <Token Name="leftBrace"
         TT="JSON_Punctuation" V="{"/>
       <Token Name="entry" TT="JSON_Entry">
        <Token Name="name"
          TT="JSON_Literal" V="pk"/>
        <Token Name="colon"
          TT="JSON_Punctuation" V=":"/>
        <Token Name="value" TT="JSON_Element">
         <Token TT="JSON_Literal" V="1"/>
        </Token>
       </Token>
       <Token Name="more" TT="List"/> (omitted)
       <Token Name="rightBrace"
         TT="JSON_Punctuation" V="}"/>
      </Token>
     </Token>
     <Token Name="rightBracket"
       TT="JSON_Punctuation" V="]"/>
    </Token>
   </Token>
  </Token>
 </Token>
</Program>
```

Figure 12: XML representation of a PST

According to the first line, this took approximately 1 ms to parse, with 52 parsing steps. There are 35 tokens in the final PST. Note that the "more" token was omitted from this listing.

9 Additional Tools

Initial versions of these code analysis tools are available.

We define a project as a (possibly large) collection of computer programs written in a variety of different programming languages. In our system, a project is a Java class that specifies what language to use for each source file, what character encoding it has, how to interpret tabs in it, etc. A project also has a very small editor built-in to repair known problems in specific files. For example, missing semicolons can be added or typos repaired. A project uses a simple regular expression matcher on pre-specified line numbers on specific files.

Traditional Grammar Generator

Given a Programmar, a traditional BNF-like grammar can be generated from it. However, since terminal nodes are expressed as Java methods, not as grammar rules, they cannot be generated automatically. When used on a collection of computer programs, frequency counts are also available showing how many instances of each rule are present in that project.

Pretty Printer

Once a computer program has been parsed, the parsing results can also be used to regenerate the original program, but in a

canonical form. Indentation can be fixed, extra spaces eliminated, capitalization standardized, etc. Elements in the Programmar can have annotations on them to assist with the output formatting, such as @NEWLINE, @NOSPACE, @INDENT, and @OUTDENT.

Programmar Debugger

The parser can generate a tracing output, either in plain text or html. Sometimes this output is not sufficient, such as when debugging a parse failure. A visual debugger is available for this with commands like Step-in, Step-over, Continue, etc.

Program Inspector

Once a program has been parsed, an html report can be created for it, with color codes for all the different kinds of terminal nodes. In addition, elements in the Programmar can be labelled with @DOC("href") to create a hyperlink to an online document describing that keyword.

Inventory Browser

While working on large projects, or more than one project, it is often useful to monitor parsing progress. Web-based tools are available to show all active projects and all active languages. For each project, all the files are shown with statistics like number of lines, size of the resulting parse output, parsing speed, etc. If a parse fails, a link is provided to view the details of the parse failure, including the lines around the failure.

For each language, details are shown as well for each source file such as parse success rates for that language and average parse speeds. A BNF-like grammar for that language is also viewable, with frequency counts.

Macro Pre-Processor

For many languages such as C, PL/I and COBOL, parsing is sometimes not possible in a single pass. We have a pre-processor available to resolve macros. Generally, not all macros need to be expanded, so controls are available to choose which macros to expand and which to leave intact.

10 Future Work

We plan to create an open-source repository for Programmars as well as an API for parsing programs over the web.

Work has already begun on connecting variable references to their definitions. In some cases, this can be done while parsing, but in many cases such connections must be done in a separate step because they depend on successfully parsing other files.

Ultimately, this work is intended to be helpful in application modernization, especially from legacy programming languages to more modern languages.

11 Conclusion

The Programmar approach builds on top of traditional parsing technologies. It greatly facilitates scalability and cross-language processing, and it is also context-sensitive (for terminal nodes). As of this writing, several dozen programming languages have Programmars built for them, with varying degrees of completeness. These Programmars have been used to parse millions of lines of code. A patent is pending on this technique.

12 References

[1] C. Preimesberger. Updating Legacy IT Systems While Mitigating Risks: 10 Best Practices, eWeek, Mar. 19, 2014, 7.

[2] R.L. Mitchell, M. Keefe, The COBOL Brain Drain, Computerworld, Vol. 46, Issue 10, May 2012, 18-25.

[3] A.J. McAllister, Automation-Enabled Code Conversion, Proceedings SERP '10: International Conference on Software Engineering Research and Practice, July 2010, 11-17.

[4] Modern Systems: COBOL Conversion and Migration, http://www.ateras.com/solutions/legacy-migration/cobol-migration.aspx

[5] Semantic Designs: COBOL Migration, http://www.semdesigns.com/Products/Services/COBOLMigration.html

[6] MSS International: COBOL to Java Conversion, http://www.mssint.com/sites/default/files/MSS-Cobol-to Java-conversion.pdf

[7] RES – A Pure Java Open Source COBOL To Java Translator, http://opencobol2java.sourceforge.net

[8] eranea: Modernize your core IT system toward Java, Web, Linux, http://www.eranea.com

[9] S.C. Johnson, Yacc: Yet Another Compiler-Compiler, AT&T Bell Laboratories, 1975.

[10] J.R. Levine, T. Mason, & D. Brown Lex & Yacc, O'Reilly & Associates, 1992.

[11] T.A.Wagner, S.L. Graham, Incremental Analysis of Real Programming Languages, Proceedings of the ACM Conference on Programming Language Design and Implementation, 1997, 31–43.

[12] D. Jackson, M. Rinard, Software analysis: A roadmap, Conference on The Future of Software Engineering, 2000, 135–145.

[13] Z.P. Fry, D. Shepherd, E. Hill, L. Pollock, K. Vijay-Shanker, Analysing source code: looking for useful verb–direct object pairs in all the right places, IET Software, Vol. 2, Issue 1, Feb. 2008, 27-36.

[14] I.R. Forman, N. Forman, Java Reflection in Action, Manning Publications, 2004.

[15] A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison Wesley, 2006.

TD-Manager: a tool for managing technical debt through integrated catalog

Lucas Borante Foganholi, Rogério Eduardo Garcia,
Danilo Medeiros Eler, Ronaldo Celso Messias Correia, Celso Olivete Junior
Faculdade de Ciências e Tecnologia, UNESP – Univ Estadual Paulista,
Presidente Prudente, Departamento de Matemática e Computação
Presidente Prudente/SP, Brazil
E-mail: borante@gmail.com, {rogerio, daniloeler, ronaldo, olivete}@fct.unesp.br

***Abstract*—Technical debt is an emergent area that has stimulated academical concern, its practical application cope development activities such as documentation, design, code and test. However, literature review pointed out an integration gap between identifying and accurately cataloging technical debt. It also mentioned bunch of tools for most activities on software development process that could identify technical debt, but there is not a described solution that supports cataloging all types of debt. In this context, this work aims to present an approach to catalog technical debt related to any activity mentioned, tabulating and managing its properties. This catalog is implemented by TD-Manager tool, that allows register technical debt and control debt status. In addition, the tool can integrate with technical debt identification tools and import debt to catalog. In order to show the approach, we present an integration of technical debt identified using Sonar, mapping relationship and managing external information integrated.**

Keywords: Technical Debt, Debt Cataloging, Integration on Debt Identification, Technical Debt Management, Technical Debt Cataloging Tool

I. Introduction

The term "technical debt" is used as a metaphor to refer to the likely long-term costs associated with software development and maintenance shortcuts taken by programmers to deliver short-term business benefits [1], [2], [3]. During software development is usual to prioritize some activities, leaving others in background, such as defect correction and documentation. The main reason for that is to reach the expectations imposed due to time or financial constraints. Thereby, postpone a technical activity creates technical debt.

Technical Debt (TD) provides useful guidance when a trade off of code quality must be made against another factor, e.g., delivering functionality more quickly to achieve time to market objectives [4]. Seaman and Guo [5] explain that software managers, to use that trade off, need to balance the benefit of incurring TD with the associated costs when planning their projects, taking decisions supported by information. Although, the lack of information is a problem.

Letouzey and Ilkiewicz [6] described some of these manager actions such as setting targets for debt and specifying what level and which debt types are acceptable for the project or organization; analyzing and understanding debt to estimate potential impact and provide rationale for decisions; using TD as input for governance of application assets and analyzing an application's debt in correlation with other information such as business value or user perceived quality; and institutionalizing the previous practices and putting in place tools and processes to produce the benefits of proactive TD management. To assist in those activities, there are techniques (such as code smells, design patterns, test results for defect debt and comparing with test plans for testing debt) and tools (such as *FindBugs*, *Sonar*, code coverage tools) that could potentially be useful in the identification of TD, even if many of them were not developed for that [7].

In this scenario an existent problem consists in catalog TD items independent of the debt type, and use contextualized information when the debt is detected. Thus, the goal of this paper is to present a tool named TD-Manager, which aims to create a catalog integrating TD from design/code, test, documentation, defect and infrastructure activities. It also intends to integrate with database of identification tools in order to import technical debt.

The methodology used in this paper consists in extending the described contents of TD items proposed by Seaman and Guo [5], creating an integrated catalog, for managing TD of any type, in a semi automatic process. TD-Manager is a proposed tool that keeps records of technical debts, as described in this paper. The integration made between TD-Manager and *Sonar* allows catalog and manage the information after TD identification accomplished in the second tool.

This paper is organized as follows: in Section II is presented a background used in the work, in Section III is detailed the proposed approach, in Section IV is described TD-Manager tool which implements the proposed approach, in Section V is demonstrated a case of study using the tool integrated with *Sonar*, in Section VI is presented a discussion about the proposed approach and in Section VII is presented the conclusion and future works.

II. Background

A. Identification of Technical Debt

Guo and Seaman [8] proposed a classification of technical debt into four main types: design (or code), testing, defect and documentation debt. In each type of TD is possible to find techniques and tools to identify the debt.

Design or code debt can be identified by statically analyzing source code or inspecting code compliance to standards [5], Izurieta et al. [7] mentioned techniques and tools that could potentially be useful in the identification activity. They presented four specific identification techniques: code smells, automatic static analysis, modularity violations and design patterns, described in the following.

Code smells (a.k.a. bad smells), which the concept was introduced by Fowler [9], describes choices in object-oriented systems that do not uses principles of good-object oriented design. Automatic approaches have been developed to identify code smells, as proposed by Marinescu [10]. Former studies have shown that some code smells are correlated with defect- and change-proneness [11]. In context of code smells technique, an example of tool is CodeVizard [11].

Automatic static analysis (ASA) is a reverse engineering technique that consists in extracting information about a program from its source code using automatic tools [12]. ASA tools search issues based on violations on recommended programming practices and potential defects that might cause faults or degrade some dimensions of software quality such as maintainability, efficiency, among others. Some ASA issues can indicate TD as they are good candidates for removal through refactoring to avoid future problems. In previous work Vetro et al. [13] analyzed the issues detected by *FindBugs* on two pools of similar small programs. Some of the issues identified as good/bad defect detectors by [13] were also found in similar studies with *FindBugs*, both in industry [14] and open source software [15]. Some studies have also been conducted with other tools (e.g. *Sonar*) and the overall finding is: a small set of ASA issues is related to defects in the software, but the set depends on the context and type of the software.

Izurieta et al. [7] described a context for modularity violations and how it connects with technical debt. During software evolution, if two components always are changed together to accommodate modifications but they belong to two separate modules designed to evolve independently, then there is an unconformity. Such unconformity can indicate TD, as they may be caused by side effects of a quick and dirty implementation, or requirements might have changed such that the original designed architecture could not be easily adapted. When such discrepancies exist, the software can deviate from its designed modular structure, which is called a modularity violation. Wong et al. [16] have demonstrated the feasibility and utility of this approach. In their experiment using *Hadoop*, they identified 231 modularity violations from 490 modification requests and 152 (65%) violations were conservatively confirmed. For modularity violations, another example of tool is *CLIO* [16].

Design patterns [17] are popular because, and the reasons are not limited to, it claims to facilitate maintainability and flexibility of designs, to reduce number of defects and faults, and to improve architectural designs. Software designs decay as systems and operational environments evolve, and it can involve design patterns. Classes that participate in design pattern realizations accumulate grime non-pattern related code. Grime represents a form of TD, since the effort to keep the patterns cleanly instantiated has been deferred. In prior studies Izurieta and Bieman [18] introduced the notion of design pattern grime and performed a study on three open-source systems, *JRefactory*, *ArgoUML* and *eXist*.

The four described techniques focus on source code or design TD type. There are other types of TD such as testing, defect and documentation, that will be contextualized in the following.

Testing debt are tests that were planned but not implemented/executed or got lost, it is also test cases not updated for new/changed functionality or low code coverage [5]. They can be identified by comparing test plans to their results, created on planning tests and not executed and, again, when identified low code coverage [5].

Most testing tools mentioned are not planned to identify TD, but it can be used in explained situation. Yang and Li [19] survey Coverage-based tools and compared 17 tools based on three features: code coverage measurement, coverage criteria and automation and reporting. Shah and Torchiano [20] present through systematic review the consequences of exploratory testing as TD. Exploratory testing is an approach that does not rely on the formal test case definition – instead of designing test cases, the execution and evaluation of the software behavior are based on tester intuition and knowledge.

Documentation debt are documentation that is not kept up-to-date [5], such as API documentation, requirements and use case documentation. They can be identified by comparing change reports to documentation version histories. If modifications are made without accompanying changes to documentation, the corresponding not updated documentation is documentation debt [5]. Forward and Lethbridge [21] identified, through a survey, tools used to deal with documentation in software projects, including automated generation of documentation.

Defect debt are known defects that are not yet fixed [5], such as low priority or severity defects due to rarely manifest or presented workarounds. They can be identified by comparing test results to change reports, the defects found and not fixed are defect debt items [5]. The tools used to find source code debts and the tools that support test executions are capable to identify defects. Snipes and Robinson [22] detailed a technique based on change control

boards (CCB) to categorize and prioritize defects supporting manager decisions to fix/defer debts based on cost-benefit analysis.

B. Cataloging Technical Debt

According to Seaman and Guo [5], the management of TD is based on a TD list, which is similar to a task backlog. The backlog contains TD items (in the following simply referred to as items), each one represents a task left undone that incurs a risk of causing future problems if not completed. Each item includes a description of what part of the system the debt item is related to, why that task needs to be done, and estimates the TD's principal and interest, as well as other attributes described in their work.

Initially, when an item is created, the principal, expected interest amount, interest standard deviation and correlations with other debt items can be estimated subjectively according to the maintainer's experience [5]. Since it is uncertain whether extra effort will be required, they used expected interest amount and standard deviation to capture the uncertainty. These rough estimation can be adjusted later using historical data or other types of program analysis.

III. The Proposed Approach

In this section is described the proposed approach which has three activity groups: *Identification*, *Cataloging* and *Managing*, as depicted in Figure 1

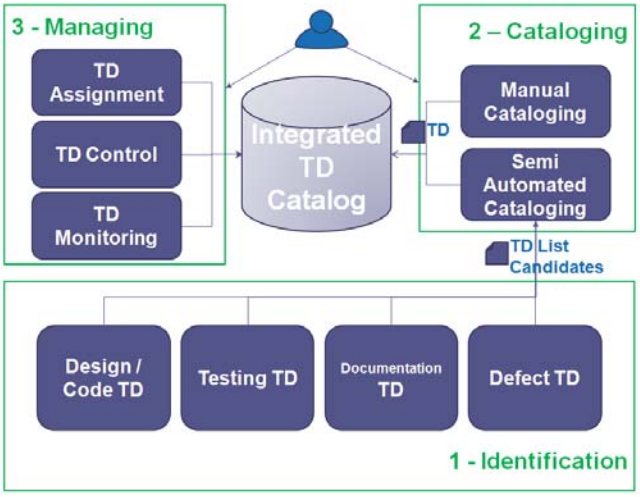


Figure 1: Proposed Approach

The *identification* group initiates the proposed approach and copes technical debt identification activities for the types design (or code), testing, documentation and defect. The result or output of this activity is a list containing candidates to TD. As exposed in the previous section, there are techniques and tools for identification of each kind of technical debt mentioned.

Technical debt *cataloging* is the second group in the proposed approach and initiates after identification. This group has two activities: manual cataloging and semi automated cataloging, explained in the following. Both activities must be performed by a user, in the role of collector. The output of *cataloging* is a set of register of technical debt.

Table I: Technical Debt Template - Adapted from [5]

ID	TD identification number
Date	Date of TD identification
Responsible	Person or role which should fix this TD item
Type	design, documentation, defect, testing, or other type of debt
Project	Name of project or software application
Location	List of files/classes/methods or documents/pages involved
Description	Describes the anomaly and possible impacts on future maintenance
Estimated Principal	How much work is required to pay off this TD item on a three point scale: High/Medium/Low
Estimated Interest Amount	How much extra work will need to be performed in the future if this TD item is not paid off now on a three point scale: High/Medium/Low
Estimated Interest Probability	How likely is it that this item, if not paid off, will cause extra work to be necessary in the future on a three point scale: High/Medium/Low
Intentional	Yes/No/Don't Know
Fixed By	Person or role who really fix this TD item
Fixed Date	Date of TD conclusion
Realized Principal	How much work was required to pay off this TD item on a three point scale: High/Medium/Low
Realized Interest Amount	How much extra work was needed to be performed if this TD item was not paid off at moment of detection, on a three point scale: High/Medium/Low

Manual cataloging is the activity to catalog any technical debt manually, independent of debt origin or way of identification it can be registered in this activity.

Semi automated cataloging uses the candidates of TD list generated in the previous group as input and, after analysis, catalog the item from the TD list. The collector analyzes the list and, for each TD item contained, reject or register as TD item, fulfilling the required information such as *Date*, *Responsible*, *Type*, *Description* and the estimation attributes, if not defined.

For both described activities of *cataloging* group is necessary specify the integrated catalog which explains the documentation structure for technical debt. The structure for technical debt catalog is based onto the item structure described in subsection II-B proposed by Seaman and Guo [5]. This structure is extended in the proposed approach in order to properly present technical debt at the management level. Item structure are described in Table I.

The last group in the proposed approach is *Managing*. It comprehends TD assignment (or reassignment), TD control and TD monitoring activities. These activities use the explained TD catalog as input and, with this information, supports manager decisions. TD assignment activity is used to perform attribution of TD to a responsible. TD control activity is used to manage TD, modifying properties, delete

or changing TD status. TD monitoring activity is used to track TDs life-cycle, due dates and, when some changes are needed, it can use previous activity to perform any modification.

IV. TD-Manager tool

TD-Manager is a tool that implements the approach (protocol) described in this paper. TD-Manager uses the integrated catalog as metadata in order to register technical debt properties. It is was developed using Java language and public community maintained frameworks to improve development productivity and reuse. The tool was developed using MVC (Model View Controller) design pattern and the frameworks used for persistence (or model) and interface (or view) layers are *Hibernate* and *Vaadin*, respectively.

It is a web application with an internal *Tomcat* as web container, allowing to execute the web application archive (war) file as a Java archive (jar), and as a web-based software the advantages as detailed in [23]. Since TD-Manager uses *Vaadin 7*, the web browsers supported by this version of the framework is also supported by the tool, but it was intensively tested using *Google Chrome* browser.

In the model layer, TD-Manager uses the proposed integrated catalog (or TD catalog) as an entity and its attributes as columns. It also has an entity for users (a.k.a responsible), one entity to register the connection information and one for mapping the relationship between TD-Manager and external applications. In the current version the tool only supports *Postgres* as its database and *MySQL, Oracle, Postgres and SQL Server* as external database.

The generic layout of the application contains a left side steady menu used to navigate among the programs (or functionalities). Each program represents one or more activity described on previous section. The menu also contains the logged user information and application logout.

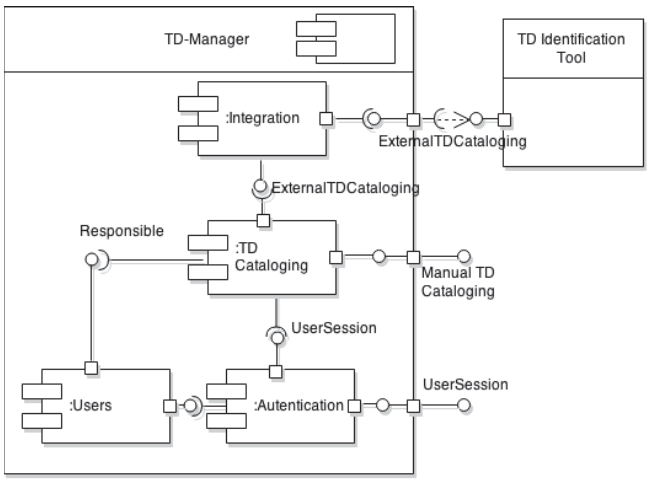


Figure 2: TD-Manager Architecture

The architecture of the solution is depicted in Figure 2 and contains *Integration*, *TD Cataloging*, *Authentication* and *User* components.

Integration correspond to external integration process, allowing to configure, test and save the information needed to connect. The first step to use this functionality is to set the connection integration properties which comprehends *Alias*, *Server*, *Port*, *Database Name*,*Database Type*, *User* and *Password*. After inform all the required fields it is possible to test the connection. This is the most difficult configuration because demands knowledge on identification tool database information and credentials, and TD-Manager, in its actual version, only allows direct database connection.

After configuring the connection it is indispensable to map a relationship between TD catalog fields and identification tool properties. For this mapping the user needs to provide external table name, external field and TD-Manager catalog field, establishing the relationship. In the end of this process the user can check results of the integration map and retrieve foreign TD candidates.

Authentication and *User* component corresponds to application authentication and permissions, associated to profiles, to access the functionalities. The user is also used when associating a TD to a responsible or fixed by person, which creates a relationship between *User* component and *TD Cataloging* component.

TD Cataloging corresponds to the register of technical debt and can be inputed manually or retrieved after integration synchronization through a foreign database connection from a TD identification tool, detailed in the following. This component also contains manager functionalities such as TD monitoring, control and assignment. It is possible trough filters manipulation, attributes sorting and status controlling.

For manual TD cataloging is only mandatory to fulfill the TD information as described in the extended catalog proposed in previous section, and no integration or extra application is used. The collector, with an external TD candidates list, can add TD after being logged into application. For this catalog it is mandatory complete the required attributes and the tool validates it, not allowing to save if they are not detailed.

For semi automatic interface it is necessary to create a database integration and mapping the relationship between TD-Manager and the external tool. The interaction between *TD Cataloging* and *Integration* component is viable only after configure connection information and create the relationship between the fields of the integrated catalog and external database table, as explained in *Integration* component. With this, it is possible to use TD-Manager interface to filter and sort the external TD identified by the integrated tool and import to TD-Manager. This external filter and sorting is a mechanism used by the controller to help on TD candidates list analysis and only the selected ones are cataloged. At this moment, is possible to override some mapped (or not, which

means null information from external identification tool) information such as responsible, TD type and the estimated fields.

In the semi automatic interface was also created an attribute to register external information ID, avoiding duplicate the cataloged TD and alerting when the external key is marked for integration twice. For the appropriate use of this functionality it is vital to proper configure the external field ID in the mapping properties. After mapping the relationship between these both tools for each property of the catalog, it is possible to check the results of configured properties (or mapping). This functionality allows the user, responsible to define the relationship, to verify external tool data in TD-Manager and change it if the information is not accurate.

In both options, it is only possible to associate any TD register with a responsible after include this person as a application user. While TD is not finished or deleted, it remains available to manager deal with TD relevance, change estimated effort to fix, debt interest or probability to cause extra work in the future. It also can control the date when TD was fixed and the realized efforts to fix it.

The implementation of integrated catalog and an overview of the tool is presented in case study.

V. Case Study Using TD-Manager and Sonar

In this case study is presented the use of TD-Manager to catalog TD from open source project named OpenRefine [24], which is a tool for working with messy data: cleaning it; transforming it from one format into another; extending it with web services; and linking it to databases. The project is being developed using Java and contains 36.883 LOCs (Lines of Code) distributed in 496 classes, a wiki as documentation [24] and some issues cataloged on *Github* repository.

Due to space constraints only one TD identification tool to perform the integration is presented. *Sonar* is a world wide adopted (ASA) tool to analyze source code as mentioned in background section. The tool is based on rules to identify source code defects or refactoring points and when integrated with *FindBugs*, in the version 3.7.4 and for Java projects, it contains a total of 516 rules. These rules are classified in severities and include bad practices, correctness and performance issues, design flaws, code issues, security, etc.

The issues are identified by *Sonar* (TD candidates) and to show the use of TD-Manager for integrate a TD catalog, it is necessary to create a connection between TD-Manager and *Sonar*. Since *Sonar* was locally installed and configured to use *Postgres* database as it issues repository, the connection in TD-Manager is created using *Postgres* database, *localhost* server and *Sonar* credentials, as presented in Figure 3. In the Figure 3 it also possible to see how TD-Manager creates the relationship, explained in the following.

After successfully test the connection, it is necessary to map the external table and fields to import TD candidates

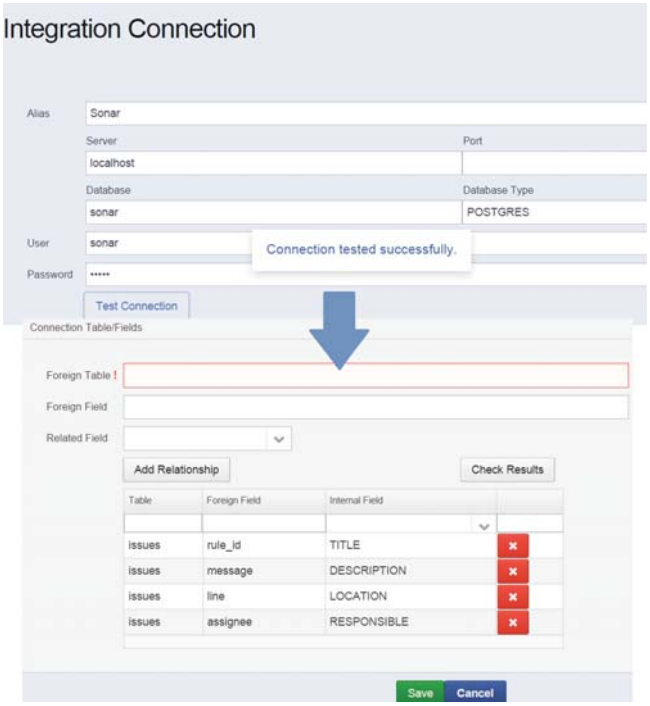


Figure 3: Mapped Integration (TD-Manager and *Sonar*)

Table II: Mapped relationship between TD-Manager and *Sonar*

TD-Manager Fields	**Sonar Fields**
Date	issue_creation_date
Responsible	assignee
Project	root_component_id
Location	component_id
Description	message

list. In this step is necessary to know the characteristic of the *Sonar*, which table resides the main information and the specific fields related to the proposed catalog. From *Postgres* database investigation on Sonar schema it is possible to find "Issues" table, which contains all identified information for *OpenRefine* project. Using the table "Issues" it is possible to create the relationship between the two application with fields as shown in Table II.

After mapping the relationship between these both tools for each property of the catalog, TD-Manager extracts the information through Integration Sync interface and the manager can filter or sort this information to register technical debt into the integrated catalog. With the information inserted into the catalog, managers is allowed to take control of it through the Technical Debt interface, which is the same to manual register TD.

In Figure 4 is presented an example of an integrated technical debt of design type assigned to the responsible named *Administrator*. The debt occurred on project OpenRefine in line 83 of class *com.google.refine.browsing.facets.ListFacet*

and was registered with low estimated principal, which means low difficult to fix, and medium interest amount, which means medium difficult if decided to postpone implementation to fix it. It also possible to retrieve information inserted after fix TD such as date when the implementation finishes, author who fixed the TD and the real effort to fix (realized principal), remaining low – same as in estimation.

VI. Discussions

It is possible to find several works related to decision making process in TD, such as [25], [26], [22]. TD-Manager tool aims to help managers to control TD and support their decision, establishing a closer relationship to those works. But besides the fact that integrated catalog is an extension of previous work [5] and includes additional information, it is possible that in the future more information for the integrated catalog is needed.

TD-Manager may cause accumulation of technical debt. The integrated catalog are designed to be complete and intuitive but it cannot be fast enough in order to capture all information of technical debts and update them. Just the needed to use another tool to manage the debt could make the responsible of the debt to not update the information.

It is being developed another work to refine and expand the approach proposed in this paper. The goal of this future work is aimed to improve an automated process of identifying technical debt and use a detailed catalog with contextualized information of identified technical debt items.

VII. Conclusions

The tool is designed to facilitate the integration between different types of TD. It is possible since unique repositories of all kind of debts are the key to group all information related to TD. Considering this design, TD-Manager focused on easily creates database integration, supporting the four most popular database engines [27], and simply assists the relationship mapping between the tools.

The approach produces and maintains a TD list according to the captured TD. TD List provides valuable information to existing software components. It can be helpful for generate agile backlog list, can provide a list of needed updates on documentation, can help on prioritize defects and can evidence faults on testing area.

It was presented a case study with one kind of technical debt (source code/design) and the tool was not yet integrated with testing, defect and documentation TD identification tools. Although, the integration is based on database direct connection, which means if any TD identification tool provides the connection credentials it is possible to integrate using the same steps described in the case study.

TD-Manager also can capture human-detected technical debt. In addition to the integration, it is possible to catalog any kind of technical debt identified by stakeholders. Since they are fully aware of all active requirements and development conventions, they can find additional technical debts, not identified by any tools. This ensures that information regarding the any kind of integration is possible to be cataloged.

By grouping technical debt identification from different project development level (code, test, and document), TD-Manager tool ensures that development is conducted while aware of technical debt's presence. That allows any stakeholder to avoid unintentionally increasing the value of technical debt through dependencies to affected areas and to decrease efficiently its value by tackling technical debt in areas where the project is currently conducted.

TD-Manager allows multiple TD indicators to be used instead of only one of the mentioned tools and it is strongly recommended, since different tools point to different problems in a software artifact. The use of a single tool or single indicator (e.g. a single code smells) will only in rare cases point to all important TD issues in a project. As a result, project teams need to make intentional decisions about which of the TD indicators are of most relevance to them, based on the quality goals of their project, as suggested in [**?**].

After apply TD-Manager on a small software development company for Java developers and tester teams, the project manager reported positive results in adoption of the tool. He stated that, besides the only three option to control estimated and realized efforts, it was possible to handily TD inventory with the necessary information to support decisions. It was also possible to compare estimated versus realized when fixing TD, and it helps on monitoring individual performance and accurately estimation of TD. With continued development, getting more people involved on technical debt scenario and its management, it is expected to discover ways to further support the projects' TD management through enhancements in the TD-Manager.

We planed TD-Manager tool to enhance productivity for industrial settings. So, we expect to further improve and validate TD-Manager in such environments. For that, we have planned an extensive series of case studies with an experimental evaluation on open source software. These studies will cope with TD management over finished and ongoing open source software projects with characteristics desirable like access to software documentation, issues repository and existence of test plans. For finished products we intend to use apply technical debt identification and assessment tools in order to simulate the life-span of technical debt.

References

[1] W. Cunningham, "Object-oriented programming systems, languages, and applications," in *The WyCash Portfolio Management System*, 1992.

[2] N. Brown, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, N. Zazworka, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, and R. Nord, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER*, 2010.

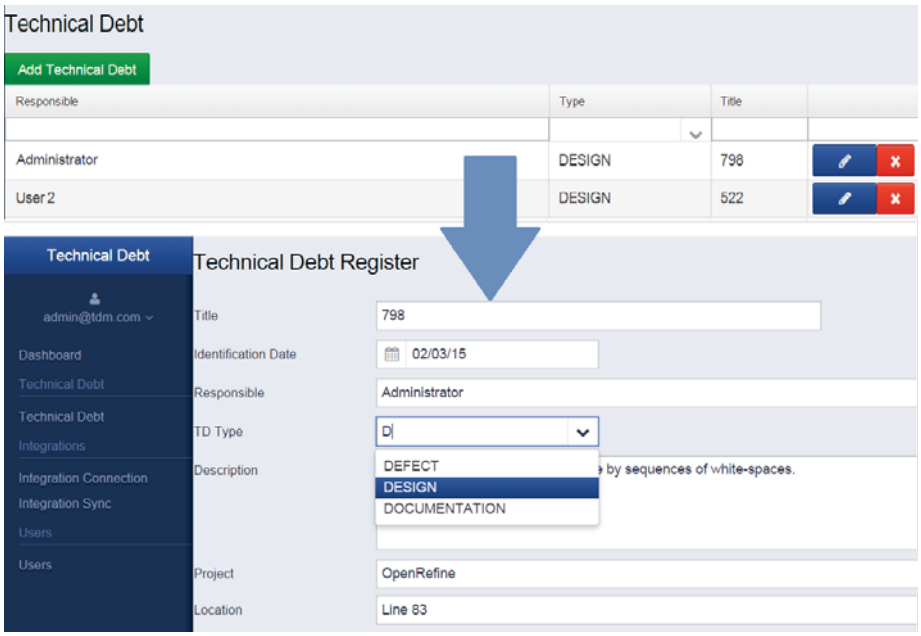


Figure 4: Implemented Integrated Catalog

[3] P. Kruchten, R. L. Nord, I. Ozkaya, and D. Falessi, "Technical debt," *ACM SIGSOFT Software Engineering Notes*, vol. 38, pp. 51–54, 2013.

[4] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *2nd working on Managing technical debt - MTD '11*, 2011.

[5] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," *Advances in Computers*, vol. 82, pp. 25–46, 2011.

[6] J. L. Letouzey and M. Ilkiewicz, "Managing technical debt with the SQALE method," *IEEE Software*, vol. 29, no. 6, pp. 44–51, 2012.

[7] C. Izurieta, A. Vetro, N. Zazworka, Y. Cai, C. Seaman, and F. Shull, "Organizing the technical debt landscape," in *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012, pp. 23–26.

[8] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceeding of the 2nd working on Managing technical debt - MTD '11*, 2011.

[9] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[10] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," *IEEE International Conference on Software Maintenance*, vol. 0, pp. 350–359, 2004.

[11] N. Zazworka, M. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality," in *22nd Workshop on Managing Technical Debt (MTD '11)*, 2011, aCM, New York, NY, USA.

[12] D. Binkley, "Source code analysis: A road map," in *FOSE '07 2007 Future of Software Engineering*, 2007, pp. 104–119.

[13] A. Vetro', M. Morisio, and M. Torchiano, "An empirical validation of findbugs issues related to defects," in *Evaluation e Assessment in Software Engineering (EASE 2011), 15th Annual Conference*, 2011.

[14] N. Ayewah and W. Pugh, "The google findbugs fixit," in *Proceedings of the 19th International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2010, pp. 241–252.

[15] S. Kim and M. Ernst, "Prioritizing warning categories by analyzing software history," in *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, May 2007, pp. 27–27.

[16] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Software Engineering (ICSE), 2011 33rd International Conference on*, May 2011, pp. 411–420.

[17] Y. G. Gueheneuc and H. Albin-Amiot, "Using design patterns and constraints to automate the detection and correction of inter-class design defects," in *Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on*, 2001, pp. 296–305.

[18] C. Izurieta and J. Bieman, "A multiple case study of design pattern decay, grime, and rot in evolving software systems," *Software Quality Journal*, pp. 289–323, 2013.

[19] Q. Yang, J. J. Li, and D. M. Weiss, "A survey of coverage-based testing tools," in *Computer Journal*, vol. 52, no. 5, 2009, pp. 589–597.

[20] S. M. A. Shah, M. Torchiano, A. Vetrò, and M. Morisio, "Exploratory testing as a source of technical debt," *IT Professional*, vol. 16, no. 3, pp. 44–51, 2014.

[21] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies," in *Proceedings of the 2002 ACM symposium on Document engineering - DocEng '02*. New York, New York, USA: ACM Press, Nov. 2002, p. 26. [Online]. Available: http://dl.acm.org/citation.cfm?id=585058.585065

[22] W. Snipes, B. Robinson, Y. Guo, and C. Seaman, "Defining the decision factors for managing defects: A technical debt perspective," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, June 2012, pp. 54–60. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84864135572&partnerID=tZOtx3y1

[23] A. Charland and B. Leroux, "Mobile application development: Web vs. native," *Commun. ACM*, vol. 54, no. 5, pp. 49–53, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1941487.1941504

[24] OpenRefine. http://openrefine.org/. Accessed: 2015-03-27.

[25] C. Seaman, Y. Guo, N. Zazworka, F. Shull, C. Izurieta, Y. Cai, and A. Vetro, "Using technical debt data in decision making: Potential decision approaches," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, June 2012, pp. 45–48. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84864136407&partnerID=tZOtx3y1

[26] K. Schmid, "A formal approach to technical debt decision making," in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, ser. QoSA13, vol. 1. New York, NY, USA: ACM, 2013, pp. 153–162.

[27] DB-Engines Ranking. http://db-engines.com/en/ranking. Accessed: 2015-03-27.

Fault detection coverage measurement using a fault imitation method for nuclear digital controller

Young-Jun Lee[1] and Jang-Soo Lee[1] and Young-Kuk Kim[2]
[1]Instrumentation and Control / Human Factors Division, Korea Atomic Energy Research Institute
989-111 Daedeok-daero, Daejeon, 305-353, Korea
[2]Department of Computer Science & Engineering
Chungnam National University, Daejeon, 305-764, Korea

Abstract - *Analog instrumentation and control systems in nuclear power plants have recently been replaced with digital systems, and thus unexpected faults are increasing. When these faults suddenly happen, a diagnostic function of the instrumentation and control system shall detect the faults and judge whether to apply a wake-up of the defense-in-depth system. There are limits to defining and confirming the unexpected faults from a traditional development process including verification and validation activities. However, it is possible to check the performance of a diagnostic function through a fault injection technology. The reliability of a nuclear safety system can be improved by increasing the fault detection ratio using an imitation of the hardware faults, the use of specific fault injection method, and a revision of the design.*

This paper explains the fault detection coverage using the specific fault injection method which is able to imitate the hardware faults in view of the special characteristics of a controller for a nuclear power plant. The controller faults are defined, the method is identified according to fault effect factors, and the results are calculated with the experimental data.

Keywords: controller for nuclear plant, fault injection, fault imitation, fault detection coverage

1 Introduction

Safety critical digital systems have been widely used in aircraft, aerospace, railway and nuclear power plant fields. It has also increased that digital computer faults and its failure have an effect on the system shutdown. A traditional strategy for guaranteeing the system safety is to thoroughly follow the development life cycle process. Development, verification and validation, safety analysis, quality assurance activities from the planning to installing phase, and reliability evaluation methods can prove the system safety features. However, for decades it has been obvious that the reliability, availability, and safety of computer systems cannot be obtained solely by a careful design, the quality assurance, or other fault avoidance techniques. The requirements related to an expected fault, its source, and its consequence also need to be reinforced since the function of the system does not operate as designed [6]. These features related to dependability are able to be evaluated using proper testing mechanisms. This paper explains the fault detection coverage using the specific fault injection method which is able to imitate the hardware faults in view of the special characteristics of a controller for a nuclear power plant. The controller faults are defined, the method is identified according to fault effect factors, and the results are calculated with experiment data

2 Fault injection techniques

Fault injection techniques can be used in both hardware and software systems to measure the fault tolerance of such a system. For hardware, faults can be injected into the simulation code, as well as into a physical pin in hardware. For software, faults can also be injected into simulation code such as in a distributed system, or into the implementation code of the software program. An experimental evaluation for hardware and software using a fault injection technique has been researched and has been used for evaluating the fault tolerant ability. A variety of fault injection techniques have been proposed. The goal of a fault injection evaluation is to request a design change if some failures are generated by any faults injected into a target system. Using a fault injection technique, we can estimate whether target system can perform the functions correctly. Figure 1 shows the basic components of a fault injection environment.

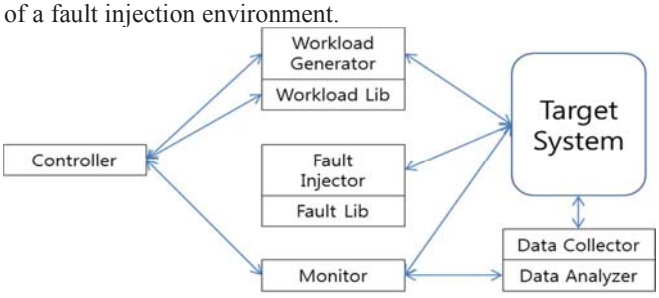


Figure 1 Fault Injection environment

2.1 Hardware fault injection techniques

A hardware fault injection technique injects the physical faults directly into the hardware system. There are several techniques: pin-level, heavy-ion radiation, power supply disturbance, and electromagnetic interference fault injection [2]-[5]. These techniques cause real damage to the target

system, and thus there exists an advantage which can evaluate the effects happening in a real system. It can also access locations that are hard to be accessed by other means. However, a specific hardware device is required to inject some faults at specific internal points, and it is hard to design a processor that has high complexity and a rapid calculation speed. The costs are more expensive when a permanent failure occurred in the experiment devices by radiation.

2.2 Software fault injection techniques

A software fault injection technique is an attractive method that can inject faults into the specific location of software. It does not need to use a high expensive hardware and excessive experimental cost. The concept of software fault injection technique is that be used to provide experimental data for evaluating the software reliability. Reliability estimation might be measured to use the methods that inject the faults into an internal point of the software system model. The goal of a software fault injection technique is to regenerate the specific faults generated in the hardware system. The regenerated faults can be represented as another type of fault like memory data modification or a mutation of the application. In contrast of hardware fault injection, a software fault injection technique has some advantages:

- Low complexity and small development efforts
- Low implementation cost
- Highly adaptation
- Easy expansion
- There are little risks that affects damage to the syste m
- No problem happened even though physical interfere nce occurs

However, there exist some disadvantages:

- It is impossible to inject faults into particular locatio ns that are inaccessible to software
- Software can inhibit a task operated in a target syste m, and can modify the original software architecture. A careful design for a fault injection environment is needed
- Error behavior in short latent time faults such as Bus or CPU defect might not be observed because of the low time resolution

To solve these disadvantages, a hybrid approach that combines two or more of the fault injection techniques to more fully exercise the system under analysis has been utilized. It is well suited for measuring extremely short latencies. However, this approach also needs a lot of cost for hardware and can decrease the flexible options by limited observation point and storage size limitation.

There are a variety of tools that are useful for injecting any faults into a target system [9][10].

3 Fault injection methodology in nuclear specific controller

3.1 Fault injection using shared memories

As explained in chapter 2.1, a method that injects faults into hardware directly needs numerous costs and it is impossible to precisely decide the fault location in terms of bits. The experimental target system might not be reused since the hardware faults can generate fatal errors at the target hardware. Numerous hardware modules such as an I/O module and communication module will also be required when thousands of experiments have been performed. Another fault injection approach is necessary because normal experiments are impossible.

The Programmable Logic Controller (PLC) used for Nuclear Power Protection Systems consists of a CPU, I/O, Communication, Power, Bus module, and so on. The I/O module sends/receives data to/from external systems. The communication module exchanges information with different channel systems through the network. The processor module operates safety application software and generates the control signal for the normal operation of nuclear power protection systems.

The PLC conducts repeated safety functions every cycle. It is different from a general software program architecture that changes the operational sequence by an interrupt action reassigning the priority of several tasks when any real event occurs in the middle of running the function.

Hardware modules send/receive required data to/from other modules using shared memories and use separate spaces in shared memories in order to communicate with other hardware. They have a self-diagnostic function, which is performed by itself. The diagnostic results are notified through the special registers in the processor module. Shared memories store the input data of external modules, calculation data, and hardware status in an internal space. We can equate the fault of shared memory and the fault of the target hardware module because the data fault in shared memory can represent the fault of an external hardware module.

This paper proposes a target-specific fault injection methodology considering the feature of the PLC since it is impossible to directly inject the faults into external modules.

Software feature of a processor module in the PLC controller are as follows:

- Data send/receive using shared memory
- Repeat function per periodic time (ex. 50ms)
- Application program and agent program

- Hardwar diagnostic function
- Hold idle status until one scan

Fault injection into shared memory can represent the injection on external hardware based on these features. Faults on the shared memory and detection ratio of injected faults can also represent the fault detection coverage of an external module.

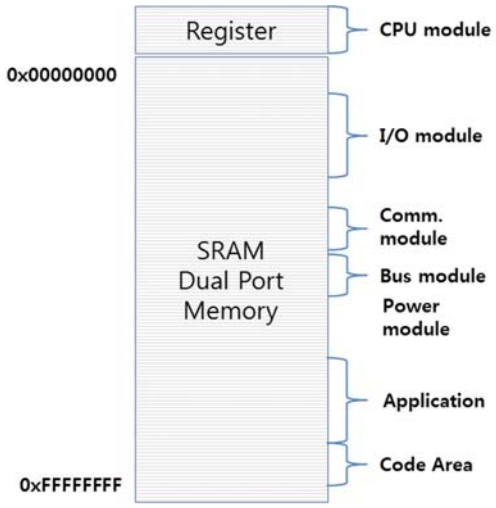


Figure 2 Relation between DPM and external module of controller

As shown in figure 2, the PLC controller of the protection system in a nuclear power plant uses the shared memories for an interface with external modules. The fault injection method into shared memory can represent a fault injection of hardware modules inside the PLC controller. This technique can overcome the disadvantage of the method directly injecting the physical faults into the target hardware.

3.2 Hardware fault imitation method using software

Hardware fault imitation method is able to slightly overcome the disadvantage that can occur when injecting physical faults into hardware thanks to the use of shared memory instead of injecting the faults directly into external modules. However, the problem of high cost does not be solved perfectly. Although shared memories are utilized for fault injection experiment, if radiation or electromagnetic waves will be irradiated into shared memories, the shared memories will be fatally damaged, so those will not be reused any more. This paper proposes and develops the hardware fault imitation device using software to solve these disadvantages [8].

The hardware fault imitation device holds fixed values, such as bit 1 or bit 0, without any change in the specific location of the hardware during operation of the software program. If there are never changes in the memory space, the memories can be considered as permanent damaged modules. It imitates a permanent hardware failure by holding the value of specific location as bit 0 or bit 1. The faults are not real hardware faults since faults injected by the imitation device can take the place of hardware faults. Thus, hardware modules are not frequently required although many experiments are continued

4 Definition of faults in digital controller

The hardware for software program is used limitedly since the system task and safety application in the controller device just have operated the same program repeatedly. So, it is necessary to define the meaning of hardware faults. The faults in the hardware do not mean the system failure since faults mean the potential source that can happen in previous phase and can cause the system failure by the results of faults. Faults might derive the system failure because of the bad effects toward the safety program and system task. Although, for example, specific location in memory was stuck at bit0, integer value used for software program might not be changed since upper one bit in Byte are always set in value 0 when the application program only uses the reserved space as integer type. When memory of integer type having space of 10 bit stores the number of 15, the binary values addressed in space of 10 bit might become ‘0000001111’. The upper 4 bits might always become value 0. Therefore the stuck-faults have no direct effect on safety software operation although upper 4 bits of byte are stuck at value0 through the external shock. These faults in digital controller device should not be calculated as factors of faults.

It is necessary to compartmentalize whether application program operated in controller device may use the specific memory space or not, in making calculation of the fault ratio for controller system. The fault happened in location that application program, running the design functions, has never utilized may be excluded from the list of faults. Detecting the faults is to be difficult in nature when real faults have no effect on safety program and it is also not easy to develop the diagnostic program that can detect the faults into an unnecessary location. Real time detection about faults may not be guaranteed in case of implementing the supervisory program with complex algorithm. The effectiveness of faults are assigned with the possibility that the faults may have an effect on the controller’s functions.

5 Test case target and result

5.1 Experiment target

A PLC used to make a nuclear reactor protection system is used as the experiment target for calculating the fault detection coverage after injecting the faults into the system. The PLC consists of a processor module, communication module, power module, bus module, and other modules. Among these modules, this experiment calculated the fault detection coverage with the CUP register of the processor module and memory for the application program. The CPU of the processor module used in experiment possess a variety of memories, in particular, the Extended Precision Register (R7~R0). The R7~R0 register supports the calculation for a 32 bit integer and 40 bit float and stores the results in it. The CPU registers are important hardware since the faults in the register might have serious effects on the calculation results. The calculation function is performed repeatedly after loading the stored data in memory into R7~R0 registers. The fault in the memory space used by the application program can be considered to simulate the faults in the application program itself. The fault detection coverage for the entire PLC can be computed by the fault injection for the registers and shared memories. This paper, however, does not perform fault injection into shared memories that have an interface to other hardware such as the I/O module, bus module, communication module, and power module. Experiment targets are only the registers in the CPU and the memories used for the application program. An additional experiment will be performed in the future.

5.2 Experiment result

The fault detection coverage is used to measure the system capability in order to detect the faults and can also be defined as a condition probability that may find intrinsic faults [7].

$$C = \Pr(\text{fault detected} \mid \text{fault existence})$$

$$= \sum FDi \,/\, \sum Fi$$

System faults can be acquired by combining the injected faults in all hardware that consist of a controller device. Individual fault detection coverage of the hardware module can be calculated using the ratio of injected faults to detected faults among them.

$$\sum F = \sum F(ram) + \sum F(io) + \ldots + \sum F(power)$$
$$\sum FD = \sum FD(ram) + \sum FD(io) + \ldots + \sum FD(power)$$

It is simply possible to get the total faults into the system by summing each fault; however, there exists a real problem in injecting the faults directly into the hardware modules.

The features of the faults in the register and memory are as follows.

[Table 1] Fault effect factors

Fault Type	Stuck-0, Stuck-1	
Duration	Permanent	
Location	0~31 bit	
Weighting	1~10	R0(0~15bit), R1(0~15bit) = 10 R0(16~31bit), R1(16~31bit) = 5 R2~R7 (0~31bit) = 1
Recovery	0	

This paper tries to quantify the faults by assigning weight values into injected faults with the results generated by injecting the stuck-fault onto all bit positions of the registers and memories. The weighting value of a fault is randomly assigned for now. It needs to be decided according to the usage frequency, and thus a more detailed and exact value will be adapted when an appropriate methodology assigning the weighing factor is established later. A failure may be determined by comparing the memory data in a normal state and memory data after injecting any faults. It may also be decided when a heartbeat signal by a system halt caused by a loss of address of the stack pointer is generated.

Reviewing the experiment, most errors of the CPU register in the processor module put the system into an endless loop state. They often generate the wrong output by injected faults into R0, R1 registers. According to the weighting value in R0, R1, the evaluation results of the pertinent registers affects the entire evaluation results of the system.

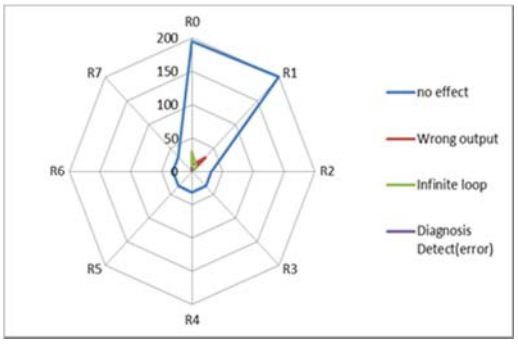


Figure 3 Fault diagnosis ratio with weight

The fault detection ratio is calculated as follows

Stuck-At-0	R0	R1	R2	R3	R4	R5	R6	R7
no effect	195	200	32	32	32	32	32	32
Wrong output	10	30	0	0	0	0	0	0
Infinite loop	30	10	0	0	0	0	0	0
Diagnosis Detect(err	5	0	0	0	0	0	0	0

There is a task that performs a diagnostic function in the processor module of a nuclear controller. It monitors the entire state of a controller device by validating whether a variety of modules perform the exact periodic action or not, checking the received signals from the hardware modules and sending a heartbeat signal to the hardware modules periodically.

It is validates the dependability of data through a periodic comparison analysis with memories utilized in the application program and backup memories.

A fault is identified by a diagnostic function implemented in the controller inside in the case of injecting random faults into the hardware module. The diagnostic probability of injected faults is as follows: 'an infinite loop state' is identified as a case of detecting the faults since the safety function is activated by a defense in depth protection system.

$$\sum FD\ (reg\ ) / \sum F\ (reg\ ) = 6.7E\text{-}02$$

A 'no effect' state can be excluded from the factors for a probability calculation based on the reason that the injected faults have no effect on the software and the safety function may be operated ordinarily. The fault detection coverage will be again calculated as 5.3E-01 except for a 'no effect' state. Fault detection coverage calculated using the same methodology in the case of injecting stuck-0 faults becomes 5.7E-01. Thus, fault detection coverage for a register injecting stuck0 and stuck1 fault can be calculated as the average of two results.

Average (stuck0, stuck1) = 5.5E-01

An experiment on the memory that the application program is operated in is also performed using the same methodology.

	fault(stuck-0)	fault(stuck-1)	App. Fault
No effect	104,790	80,050	184,840
Infinite Loop (Watchdog timer detect)	6,770	17,895	24,665
Error (but Diagnosis Function Detect)	185	65	250
Warning (but Diagnosis Function Detect)	210	405	615
Wrong Output (Trip Signal Error)	6,305	19,985	26,290
Application Delete	300	160	460

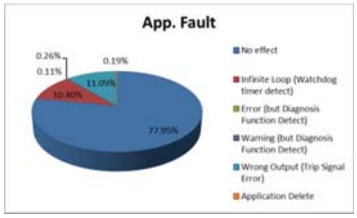


Figure 4 Experiment result of memory

Fault detection coverage for memory based on the experiment results can be calculated as 4.9E-01.

6 Conclusions

To ensure the reliability of a system, we try to calculate the reliability using a new method different from methods used traditionally. Injecting a fault in hardware, rather than a step-up action with a life-cycle process, can calculate the fault coverage of the system for detecting a failure. If such a fault detection rate is calculated, the system can improve the diagnostic capability and the design of the system is able to be enhanced. We tried to define the failure in order to calculate the failure detection coverage and attempted to differentiate for a failure with an analysis of the effect of such a failure factor. To support these claims, we have carried out tests on the registers in the CPU modules and memories for an application program of a controller device for use in nuclear power plants as the test targets.

In a later test, we will show that a failure of the shared memory can take the place of a failure of the associated hardware, and detection coverage about the injected faults into internal shared memory can be used as an element for calculating the fault coverage of the entire controller system. We can make it calculate the fault detection coverage through fault detection experiments on registers and some memory. Weighting values of the elements defined by the influence factor of a malfunction were given a different weight value depending on the frequency of use. For the fault to derive an endless loop state among the injected faults, the system will fail safely according to the diversity protection strategy. A serious failure will be derived, however, in the case of outputting an incorrect result, and thus the responsiveness of the error will be upgraded according to design improvement. It is possible to enhance the design for a diagnostic function of the controller system according to the calculation of the fault detection coverage. An enhancement of the fault detection coverage ratio through design improvements can increase the reliability probability of the control unit itself and will also be able to improve the reliability of the entire defense in depth system.

7 References

[1] M-C. Hsueh, T. K. Tsai, R.KIyer, "Fault Injection Techniques and Tools", IEEE Computer, Vol. 30, No.4, April 1997,, pp. 75-82.

[2] J. arlat et al., "Fault Injection for Dependability Validation: A Methodology and Some Applications", IEEE Trans. On Soft. Eng., vol 16, no.2, pp.166-182, Feb 1990.

[3] J.Karlsson, P. Liden, P.Dahlgren, R. Johnsson, and U. Gunneflo, "Using Heavy-ion Radiation to Validate Fault-Handling Mechanisms", in IEEE Micro, vol. 14, no. 1, pp.8-32, 1994.

[4] G. Choi, R. Iyer, "Focus: An Experimental Environment for Fault Sensitivity Analysis", IEEE Trans. On Computers, vol.41, no.12, December 1992

[5] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, J.Reisinger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", in Proc. 5th Working Conference on Dependable Computing for Critical Applications, pp. 150-151.

[6] Yu Y., "A perspective on the state of Research on Fault injection techniques," Research Report, University of Virginia, May 2001.

[7] J. B. Dugan, K. S. Ttrivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," IEEE Transactions on Computer, Vol. 38, No. 6, pp. 77-87 (1989)

[8] PATENT, "Fault mode apparatus and method using software", 2013

[9] Aidemark, J., Vinter, J., Folkesson, P., Karlsson, J.:GOOFI: generic object-oriented fault injection tool. In: Proceedings of International Conference on Dependable Systems and Networks (2001), Gothenburg, Sweden, July 2001.

[10] Carreira J., Madeira H., and Silva J., "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," IEEE Transactions on Software Engineering, vol. 24, no. 2, pp. 125-136, February 1998.

[11] Madeira, H., Costa, D., Vieira, M.:On the emulation of software faults by software faults by software fault injection. In:Proceedings of International Conference on Dependable Systems and Networks (2000)

[12] Duraes, J., Madeira, H.:Emulation of software faults: a field data study and a practical approach. IEEE Trans. Softw. Eng. 32(11), 849-867 (2006)

Validation of User Interface Model: a Systematic Literature Review

Livia Cristina Gabos Martins and Rogério Eduardo Garcia
Department of Mathematics and Computer Science
Faculty of Science and Technology
São Paulo State University – UNESP
Roberto Simonsen Street, 305
CEP 19060-900 Presidente Prudente-SP, Brazil
E-mail: liviagabos@gmail.com, rogerio@fct.unesp.br

***Abstract*—Models have been used along software development process. Their utilization to user interface (UI) development aims to deal with dependency of technology, facilitating the understanding of functionalities and behavior, for example. Besides, models have been used to support test activities of UI. However, as usual to any software artifact, we need to assure the confidence of UI models, considering their particularities. In this paper we focused on searching about *Which techniques and methods have been used to validate UI models?* We are aware of the risk at handling a broad research question. So, we conducted two systematic literature review (SLR). The first one focusing on models (*Which models are used for modeling UI?*), and the second one focusing on validation (*Which techniques and methods are used to validate the models?*). The results obtained from first SLR were used as parameters to the second SLR. Both the protocols and lessons learned are presented in this paper. Also, we discuss open issues on validating UI models.**

Keywords: Validation of user interface model, User interface model, Model validation, Systematic Literature Review, Software Engineering, Software development

I. Introduction

User Interface plays an important role in end-user systems, since it allows users to interact with systems [1]. The process of UI development has been beneficed by using models. An example of benefit is to facilitate the communication between people with different knowledge and skills, using a common vocabulary [2]. Considering that an UI is highly dependent of technology (platform or devices like smart phone, personal computer, etc.), using one model it is possible to obtain UI variations [3]. For example, Model-driven UI generation and Model-Based Design of User Interfaces (MBUI) are two methodology that allow creating a concrete UI by applying model transformations techniques [2], [4].

Also, models have an important role in validation. They help to test complex UI structure and event sequences. Hauptmann et al. [5] affirm that the UI test is easier if performed by persons, because their experience related to software and their intuition facilitate the interpretation of models high level description. However, this conventional method, who a person participate in validation process, do not cover the whole UI [6]. Automatic approaches can be used to validate the UI, but this validation process have problems. Any modification in the UI affects the test script. It is possible to separate test script of UI details by using models in test. UI details can be modifications of components between versions, different components position or details of input data in an UI. It would be not possible to reuse the script with details [5], [6].

In order to use models to validate UI, the first step is validate the models. Consistency between models is essential – it is necessary to assure consistency and quality of models. According to Trollmann et al. [7], there are two types of consistency. The first one is intra-model consistency – it concerns whether the model is correct considering syntactic and semantic rules of the language [8]. The second one is inter-model consistency – it concerns whether the model must have coherence with other models. According to Cruz et al. [8], "model validation is more difficult to assess, because one can never be sure if the model correctly captures the users requirements."

Regarding UI model validation, a SLR was conducted to point out which method has been used in its validate process. This paper summarizes the SLR conducted about how the UI models are validated in the development process. The main question is made to find this information is *Which techniques and methods have been used to validate UI models?* It is a broad research question and, consequently, we conducted two SLR. The first one focuses on model used to represent UI. The results were used as parameters to the second SLR, which focused on validation techniques. Both SLRs provided information about validation process, in a complementary way.

The remainder of this paper is organized as follows. Section II describes related works selected from literature about user interface validation. Section III presents the systematic literature reviews, their protocols, criteria and results. Section V shows lessons learned and gives insights about UI validation. Section IV presents the threats related to

the conduct of this research. Section VI provides conclusions and directions for future works.

II. Related Works

Kull [9] presents a SLR about generating graphical user interfaces (GUI) model automatically by reverse engineering. There is no citation about the model validation papers cited by [9] and no one demonstrated necessity of model validation.

Casteleyn et al. [10] shown a SLR focusing on the Rich Internet Applications (RIAs). They present interesting information about RIAs and its history, but there if few information about UI characteristic in selected papers. The paradigm related a RIAs development process is regarding to the use of models. However, the models are used to several types of representation and the complex structure of the RIA interface can be represented with more than one type of model.

Banerjee et al. [11] present a SLR about *GUI Testing*. The most papers (72 of 136 papers) used models to test generation. The most commons types of models are *event flow graphs* and *finite state machines*. The pointed out that models or specifications are used to formal verification (13 of 136 papers) and manual verification (13 of 136 papers) was used to verify the correctness of the output of a test case. However, there is no citation about the model validation process.

The models should be validated before their use [12]. Leopold et al. [13] explain five techniques to model validation. The *prototyping* approach uses the implementation of the model to collect feedback from experts. The *abstraction and filtering* approach is used to reduce the information provided to the user and to present the model with an adequate level of abstraction to experts. *Visualization of specification* is used for creating scenarios in graphical view for visualization of requirements. *Property checking* approach aims to compare models regarding their formal property, using ontologies or user questions. *Language generation* approach generates the models in natural language, to facilitate the communication between persons with different knowledge and background.

Leopold et al. use a natural language to conduct model validation. They argue that some persons who participate of software development do not have enough familiarity regarding to model representation. However, this fact is contrary to affirmation from Raneburger et al. [2], which the models support compression of information using a high level of abstraction through a symbology.

The techniques listed by Leopold et al. were used to compare papers obtained with SLR.

III. Systematic Literature Reviews

Two SLR were conducted. The first one focused on validation process of UI models, presented in Section III-A. The restricted goal has became impossible to obtain reasonable results. However, the partial results (obtained to secondary questions) were used as parameters to second SLR. The second SLR was conducted using models from first SLR as parameter aiming at model validation process. The second SLR is presented in Section III-B.

A. First systematic literature review

The goal of first SLR is to identify validation process to UI models. The search questions were divided in main and secondary questions. The main question is: *Which techniques and methods have been used to validate UI model?*.

The terms "*GUI model*" and "*User interface model*" were used to create search string about types of UI models. The "*GUI*" was used to capture desktop user interface. The term "*testing*" was used to compose other terms because models can be used in UI test (i.e. "*GUI testing*" and "*user interface testing*"). Derivations of terms "*model validation*" and "*user interface validation*" were used to expand the results. The term "*consistency*" was used as synonymous to validation.

The secondary questions are: i) *Which models are used for modeling UI?*, to know which graphic representations have been used (statecharts, graphs, etc.). The term "*behavior*" was included to represent the UI behavior. Posteriorly, the papers were read for selection using inclusion and exclusion criteria. ii) *Which approach (static or dynamic) has been used to model UI?*, to know whether the information represented in the model are static (source code, for example) or dynamic (the interaction with the UI). iii) *How to validate the UI models using a specification?*, to verify whether the technique used to validate the model takes into account the system specification. iv) *What methods or tools have been used to extract the UI model?*, to verify which tools have been used to obtain the model from UI. v) *How the models are shown to the users?*, to know how the models have been presented to users in order to be validated (i.e. using tools or using more than one model in a complementary way, etc.).

1) Search String: The main challenge to find relevant results is restrict the search string. The search string was structured into two main parts. The first part is related to the UI and the second part is related to the validation process. The research result by repositories is presented in Table I and the final search string is:

("GUI testing" OR "user interface testing" OR "GUI model" OR "user interface model") AND ("GUI validation" OR "user interface validation" OR "validation model" OR "Model validation" OR "model validate" OR "validated model" OR "model consistency" OR "behavior model" OR "behavior models").

The range of years of research has established to last 4 years, from 2010 to 2014.

2) Methodology for selection: All papers were considered for reading and selecting. The exclusion criteria are: i) Use the behavior model to observe user interaction with

Table I: Results of first SLR by repositories

Repository	Amount of Papers
ACM Digital Library	12
IEEE Xplore	22
Science Direct	12
Scopus	18
Total	64

UI, without analyzing UI model. ii) Do not present any characteristic about the models.

To the selection process, initially the abstract and the introduction of 64 papers obtained were read (Section III-A1) to observe the exclusion criteria. The most relevant papers were carefully read in order to identify how model was used, also considering the exclusion criteria. An example of selected paper is [14], in its introduction is written: "*This paper presents a new feedback-based technique to automated testing of graphical user interfaces (GUIs)*". In this case a UI model was used for testing its implementation. On introduction there is "*The smoke suite is executed on the GUI using an automatic test case replayer. During test execution, the runtime state of GUI widgets is collected*", showing that which the UI model presents states from UI.

An example of exclusion criteria is [7]. Part of the goal in the introduction is: "*(...) it is shown how the description can be used to define internal consistency as well as consistency between models*" – the goal not focus on UI model. They just used the UI model as an example about description.

Of 64 papers, only 11 were selected according to selection criteria and the results are presented in Table II. Finally, the selected papers were read trying to answer the research questions – presented in the beginning of this section.

Table II: Papers selected

Repository	Author
ACM Digital Library	[15], [16], [17]
IEEE Xplore	[14], [18], [19], [20] , [21], [22]
Science Direct	[23]
Scopus	[24]

3) SLR Results: After analyzing 11 papers, it was observed there is few information about validation process of UI model. Answering the main question only 3 papers mentioned the necessity of model validation with the requirements and/or users [15], [18], [24]. None of them presented the validation process in their work and no correlation among papers could be done. To validate UI model, Grilo et al. [18] affirm that building the model with different views, can help validation process. However, there is no citation about validation of UI model in [18]. Gupta and Surve [15] have quoted about types of validation, but there is no citation about application of validation models. The idea regarding the Hennig et al. [24] is create intra and inter model validation using the CAP3 and Movisa. First, the model is created by using the modeling language CAP3, refined by a expert and transformed to the Movisa. The UI is created based on model and validated by user and expert.

Answering the question (i) (*Which models are used for modeling UI?*), the models used are statecharts [25], [19], state machines [16], [20], [21], Concrete User Interface (CUI) [22] and event flow graph [26], [23]. The most part of models were used to analysis of UI states. In Gupta and Surve [15] and from Aho et al. [20], additional information of states of the UI events are also extracted from navigational information [18], structural [14], [17] or interface behavior [16], [17]. Lehmann et al. [17] does not explain which representation is used.

Answering the question (ii) (*Which approach (static or dynamic) are used to model UI?*), most of papers (7 of 11 analyzed) used a dynamic approach to create UI models [15], [14], [18], [19], [20], [21], [22]. Only [23] used both static and dynamic approaches to observe what had influenced the test cases generation.

Answering the question (iv) (*What methods or tools are used to extract the UI model?*), in selected papers were used the tools *GUITAR GUI* [14], *GuiDriver* [20], [21], *GUI-Tester* [23], *Eclipse Modeling Framework* [17]. In Hennig et al. [24] was used a combination of *CAP3* tool and Epsilon, and in Grilo et al. [18] was used a combination of *Spec#* tool and *REGUI model*. In Duan et al. [19] and Ramon et al. [22] were used *UsiResourcer* to make the re-engineering process. Gupta and Surve [15] used the tools *Sahi*, *Selenium*, *Framework Robot* and *Microsoft Excel*.

The consequence of using a broad main question can be observed in the answers (not) obtained about the purpose of SLR. The questions (i) (*What types of models are used to model UI?*) and (ii) (*Which approach (static or dynamic) are used to model UI?*) were answered. But the question (iv) (*What methods or tools are used to extract the UI model?*) is partly answered, because the selected papers do not have citation about the methods used. The questions (iii) (*How validate the models using a specification?*) and (v) (*How the models are shown to the users?*) were not answered. Considering that only 3 of 11 selected papers [15], [18], [24] have mentioned the validation process, we do not consider the validation process pointed out in those papers as significant.

One may note that secondary questions were useful to obtain relevant informations about UI characteristics, but not about validation. In order to focus on model validation process, the types of models obtained in this SLR were used as parameter in a second SLR, presented in the following.

B. Second Systematic Literature Review

The second SLR focuses on model validation process in Software Engineering, using the statecharts, state machines

and event flow graph as parameter to the type of modeling. The type of model Concrete User Interface (CUI) was not selected, because it is related to only the UI and the restriction about the UI models was not used.

The main question is similar to the first research, however the secondary questions are more specific. The main question is *Which techniques and methods are used to validate the models?*. The secondary questions are: i) *Which types of model are used?*, which type of model (statecharts, state machines and event flow graph) is used ii) *How the validation process uses the specification?*, this question is to know whether the specification is important for validation process iii) *Validation process regarding the user or other expert?*, this question is to know whether as expert person is relevant to validation process.

1) Search String: The types of models (*statecharts, state machines* and *event flow graph*) and derivations of *technique model validation* were used to create search string. The search string is:

(“technique model validation" OR “technique to model validation" OR “model validation methods" OR “evaluation of model validation" OR “model validation") AND (“statecharts" OR “state machine" OR “event flow graph") AND (“software engineering").

The same range of years was used, from 2010 to 2014. The results are presented in Table III.

Table III: Results of second SLR by repositories

Repository	Amount of Papers
ACM Digital Library	58
IEEE Xplore	45
Science Direct	35
Scopus	40
Total	178

2) Methodology for selection: The inclusion criteria was: the paper must explain the steps to model validation, related to the specification and user interaction. It was not used any restriction about the description and the extracted of model.

The selection process was the same of first SLR. Initially, 178 papers obtained were read (the abstract and the introduction) to identified the goal and the type of model used. The relevant papers were carefully read searching for any description about the validation process. An example of paper not selected is [27]. They used finite state machine and we found description about the validation process, however the paper does not focus on using a person in validation process (secondary question iii). A quote of paper [27] about the model validation: “*Validation usually includes checking whether the model program can execute any traces that are known to be allowed, and cannot execute others that are known to be forbidden*".

Of 178 papers, only 8 were selected to be read in detail. They are listed in Table IV.

Table IV: Papers selected

Repository	Author
ACM Digital Library	[28], [29]
IEEE Xplore	[30], [31], [32], [33]
Science Direct	[34], [35]

3) SLR Results: Related to the question (i) (*Which types of model are used?*), all the select papers are using or state machines or using the finite state machines [35], or extensions of them [28] to represent the model. Few papers using statecharts were found, but they were not selected to the detailed reading because the inclusion criteria.

The tools used were: radCHECK [28], SAL (Symbolic Analysis Laboratory) [33], FTOS [31], SDA (Solution Decision Advisor) [35], CoreASM [34] and RSA (Rational Software Architect) [29]. The frameworks used were: GEMDE (Generic Executable Model-Driven Engineering) [32] and CoDES [30].

Regarding the question (ii) (*How is the validation process using the specification?*), the specification was used to create an initial model ([28], [34]) and to compare to other models ([31], [35]). In other selected papers there is no citation about the use of specification.

Regarding the question (iii) (*How is the validation process with the user or other expert?*), 3 of 8 papers ([29], [28], [33]) have a condition which the users should have a prior knowledge about the system and the tool used for helping the validation process. However, in those papers there is no indication about the experimental process employed. In the papers [29], [28] and [33] there is no indication about the experience with computers and knowledge about the system (profile) of the validator. In the papers [33] and [28], the designers validated the model, not the final user, neither any stakeholder. In the paper [29] was the railway experts (related to project European Train Control System – ETCS) who collaborated to validate the model. In that paper, they used a CNL (Controlled Natural Language) and UML diagrams to perform the validations. But, the railway experts were trained to be able to understand and to interact with the model and language using RSA tools.

In [29] were used approaches to consistency check, scenario compatibility, property checking and specification visualization such approaches are related to Leopold et al. [13]. Di Guglielmo et al. [28] used the property checking and specification visualization approaches. Dutertre et al. [33] used specification visualization approaches. In [31], [32] and [34] there are not indication about the profile of person who validated the model and do not have information about the interaction of validators. They used the specification visualization approach to validate the models. In [35] several people (different profiles) were involved to validate the model in different phases of project. They used the

specification visualization approach. In [30] was used the specification visualization and the property checking related to an ontology. They used experts to validate the model.

IV. Threats to validity

There are three threats in the first SLR. Since the main question is broad, it was not possible to answer both main and secondary questions. It was difficulty to create the search string using key terms. Kitchenham et al. [36] suggested to do a previous research, mapping the information related to main objective, in order to identify relevant terms to research. We have taken in to account they suggestion, and several researches are made to identify the better terms. An example of search string related to this previous research is "*((Abstract:"GUI") or (Abstract:"graphical user interfaces") or (Abstract:"user interface")) and ("GUI testing" or "Gui validation" or "user interface validation" or "Automatic GUI Testing" or "Automatic user interface testing" or "Automated GUI Testing" or "Automated user interface testing" or "GUI model" or "user interface model")*". This search string resulted in 1018 papers, but there were many false positives (many papers not related to the use of UI models). The search in abstract was very restrictive, covering papers about UI, but no papers that use UI model. So, that restriction was not used.

The second threat is related to the formal research protocol. Only qualitative evaluation was considered on selecting papers (inclusion and exclusion criteria). Also, there were only two reviewers to conduct the SLR: one to review the protocol and other to analyze and to select papers.

The third threat was the use of terms related to types of models "*behavior*". Most of papers which has this term are not related to the interface behavior, but with respect to the user behavior, what generated false positives. Other terms relating to the types of models, such as navigation and tasks, could also be added to the string to be equal a "*behavior*" or this term should be removed to not generate false positives as a result.

Related to the second SLR, unlikely the first SLR, were found most relevant papers to answer the main question. The terms used to model validation as "*property checking*" could be added to the search string to generate fewer false positives.

Even using relevant repositories to conduct the research, the restriction to only four of them can be considered an external threat, because other repositories was not used. However, the search in other repositories, like CiteSeerX, returned few results and with little relevance. So, the threat was mitigated.

The restriction about number of terms in one repositories has been identified as external threat. It was necessary to reduce the search string up 15 terms. however, later it was solved with the final search string defined.

V. Lessons Learned

A broad research question was proposed to the first SLR (about validation process in UI model). The question was proposed considering the hypothesis: the papers show methods or techniques to validate UI models. However, most papers do not show UI models validation. Such fact is evidenced in selected papers presented in Related Works (Section II) [9], [10], [11] – they do not present model validation either.

To find relevant results about methods and techniques for validation, a second SLR was conducted focusing on validation process, using models found in the first SLR. However, there is no standard for validation with the type of model used. The choice of technique for model validation is related to the necessity and resources of project and the profile of person who will validate the model.

As next steps to this research about UI model validation, we can indicate two steps. The first one would be to research all techniques about model validation that can be applied to the UI models and perform a more specific search. After that, conduct a new research to figure out which different technique can be applied in UI models. The results about these researches would be relevant and would minimize the list of papers resulted for reading. The second way to find validation process is create a search string with more parameters related to all questions. However in repositories, as IEEE, there is a limitation of parameters to be used. Therefore, we are not able to do it.

VI. Conclusion

The goal of this paper is to identify the validation of methods used to UI models, using a SLR. The others SLRs found related to UI and models do not present aspects about the reliability of models and how the validation process are made in its papers selected.

Two researches were conducted to explore the subject proposed. The first one is related to the UI models and how UI model validation is presented. Only three papers mentioned the need of users in validation process, however they do not explain how the validation can be performed. Most of papers do not have mentioned about model validation process.

The second SLR was performed using the types of models found in the first SLR as parameters. The goal of second SLR was find more relevant results about types of models (*statecharts, state machines* and *event flow graph*) and how these models are validated in the software development process.

In the second SLR, few papers provide a description about methods used in validation process, mainly participation of users. It was possible to observe in the second SLR the concern with interaction of persons with models and tools. Few papers showed difficulties in the interaction of users with tools. Few papers showed how the training was offered to users in order to understand models and tools [29]. Most

of papers do not have any citation about validators profile, and no citation about selection process of validators.

Also, it was possible to classify the type of techniques applied, especially in Leopold et al. [13]. Few of those papers used one or more approaches to validate the models. Most of them use the specification visualization approach.

We observed that validators should have former knowledge about the models and they should know about the tools used to handle the models during validation process. Also, the models should be validated by validators with different profiles, according to complexity of system under evaluation. Automatics techniques can be applied to validate models in a complementary way.

Regarding the SLR research process, we used a qualitative evaluation to select papers. We are aware that quantitative evaluation criteria would facilitate the replication of SLR by other researchers. We intend to establish a formal protocol using quantitative evaluation in order to both mitigate the threat of construction and facilitate the replication by other researchers.

REFERENCES

[1] F. Gao, L. Zhao, and C. Liu, "Gui testing techniques based on event interactive graph tree model," in *Information and Automation (ICIA), 2010 IEEE International Conference on*, IEEE, Ed. IEEE, 2010, pp. 823–827.

[2] D. Raneburger, R. Popp, and J. Vanderdonckt, "An automated layout approach for model-driven WIMP-UI generation," in *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12*, 2012, p. 91.

[3] A. Pleuss, S. Wollny, and G. Botterweck, "Model-driven development and evolution of customized user interfaces," in *EICS 2013 - Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2013, pp. 13–22.

[4] V. Genaro Motti, D. Raggett, S. Van Cauwelaert, and J. Vanderdonckt, "Simplifying the development of cross-platform web user interfaces by collaborative model-based design," in *Proceedings of the 31st ACM international conference on Design of communication*, ACM, Ed., 2013, pp. 55–64.

[5] B. Hauptmann, M. Junker, and J. M. Hauptmann B., "Utilizing user interface models for automated instantiation and execution of system tests," in *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, ACM, Ed., 2011, pp. 8–15.

[6] O. El Ariss, D. Xu, S. Dandey, B. Vender, P. McClean, and B. Slator, "A systematic capture and replay strategy for testing complex gui based java applications," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*. IEEE, 2010, pp. 1038–1043.

[7] F. Trollmann, M. Blumendorf, V. Schwartze, and S. Albayrak, "Formalizing model consistency based on the abstract syntax," in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '11*, 2011, p. 79.

[8] A. M. R. da Cruz and J. P. Faria, "Automatic generation of interactive prototypes for domain model validation." in *International Joint conference on Software Technologies - ICSOFT (SE/MUSE/GSDCA)*, 2008, pp. 206–213.

[9] A. Kull, "Automatic gui model generation: State of the art," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 207–212.

[10] S. Casteleyn, I. Garrigós, and J.-N. Mazón, "Ten years of Rich Internet Applications: a systematic mapping study, and beyond," *ACM Transactions on the Web*, vol. 8, no. 3, pp. 1–46, 2014.

[11] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," *Information and Software Technology*, vol. 55, no. 10, pp. 1679–1694, 2013.

[12] P. Arcaini, A. Gargantini, and P. Vavassori, "Validation of models and tests for constrained combinatorial interaction testing," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE. IEEE, 2014, pp. 98–107.

[13] H. Leopold, J. Mendling, and A. Polyvyanyy, "Supporting process model validation through natural language generation," vol. 40, no. 8, pp. 818–840, Aug 2014.

[14] X. Yuan and A. M. Memon, "Generating Event Sequence-Based Test Cases Using GUI Runtime State Feedback," in *Software Engineering, IEEE Transactions on*, vol. 36, no. 1, 2010, pp. 81–95.

[15] P. Gupta and P. Surve, "Model Based Approach to Assist Test Case Creation, Execution, and Maintenance for Test Automation," in *Proceedings of the First International Workshop on End-to-End Test Script Engineering*. ACM Press, 2011, pp. 1–7.

[16] K. Chuang, C. Shih, and S. Hung, "User behavior augmented software testing for user-centered GUI," in *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, 2011, pp. 200–208.

[17] G. Lehmann, M. Blumendorf, and S. Albayrak, "Development of Context-Adaptive Applications on the Basis of Runtime User Interface Models," in *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '10*, 2010, pp. 309–314.

[18] A. M. Grilo, A. C. Paiva, and J. P. Faria, "Reverse engineering of GUI models for testing," in *2010 5th Iberian Conference on Information Systems and Technologies (CISTI)*, IEEE, Ed., 2010, pp. 1–6.

[19] L. Duan, A. Hofer, and H. Hussmann, "Model-Based Testing of Infotainment Systems on the Basis of a Graphical Human-Machine Interface," in *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, IEE, Ed., 2010, pp. 5–9.

[20] P. Aho, N. Menz, T. Räty, and I. Schieferdecker, "Automated Java GUI Modeling for Model-Based Testing Purposes," in *2011 Eighth International Conference on Information Technology: New Generations (ITNG)*, IEEE, Ed. Ieee, Apr. 2011, pp. 268–273.

[21] P. Aho, N. Menz, and T. Raty, "Enhancing generated Java GUI models with valid test data," in *Open Systems (ICOS), 2011 IEEE Conference on*, IEEE, Ed., 2011, pp. 310–315.

[22] O. S. Ramon, J. Vanderdonckt, and J. G. Molina, "Re-engineering graphical user interfaces from their resource files with UsiResourcer," in *Proceedings - International Conference on Research Challenges in Information Science*, 2013, pp. 1–12.

[23] G. Bae, G. Rothermel, and D.-H. Bae, "Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study," *Journal of Systems and Software*, vol. 97, pp. 15–46, Nov. 2014.

[24] S. Hennig, J. Van Den Bergh, K. Luyten, and A. Braune, "User driven evolution of user interface models - The FLEPR approach," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6948 LNCS, 2011, pp. 610–627.

[25] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231 – 274, 1987.

[26] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical GUI test case generation using automated planning," vol. 27, no. 2, pp. 144–155, 2001.

[27] M. Veanes and J. Jacky, "Composing model programs for analysis," *The Journal of Logic and Algebraic Programming*, vol. 79, no. 7, pp. 467–482, 2010.

[28] G. Di Guglielmo, M. Fujita, L. Di Guglielmo, F. Fummi, G. Pravadelli, C. Marconcini, and A. Foltinek, "Model-driven design and validation of embedded software," in *Proceedings of the 6th International Workshop on Automation of Software Test*, ACM, Ed., 2011, pp. 98–104.

[29] a. Chiappini, a. Cimatti, L. Macchi, O. Rebollo, M. Roveri, a. Susi, S. Tonetta, and B. Vittorini, "Formalization and validation of a subset of the European Train Control System," in *2010 ACM/IEEE 32nd*

International Conference on Software Engineering, IEEE, Ed., vol. 2, 2010, pp. 109–118.

[30] C. Szabo and Y. M. Teo, "On validation of semantic composability in data-driven simulation," in *Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2010, pp. 73–80.

[31] C. Buckl, D. Sojer, and A. Knoll, "FTOS: Model-driven development of fault-tolerant automation systems," in *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010*, 2010, pp. 1–8.

[32] A. Noguero and H. Espinoza, "A generic executable framework for model-driven engineering," in *2012 7th Iberian Conference on Information Systems and Technologies (CISTI)*, IEEE, Ed. IEEE, 2012, pp. 1–6.

[33] B. Dutertre, A. Easwaran, B. Hall, and W. Steiner, "Model-based analysis of timed-triggered ethernet," in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, 2012, pp. 1–11.

[34] R. Farahbod, V. Gervasi, and U. Glässer, "Executable formal specifications of complex distributed systems with CoreASM," *Science of Computer Programming*, vol. 79, pp. 23–38, Jan. 2014.

[35] O. Zimmermann, C. Miksovic, and J. M. Küster, "Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2014–2033, 2012.

[36] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering version 2.3," Keele University and University of Durham, Tech. Rep. Technical report EBSE-2007-01, July 2007.

Designing Context Sensitive Mobile User Interface

Mao Zheng[1], Olga Ormandjieva[2], and Hao Fan[3]
[1]Department of Computer Science, University of Wisconsin-La Crosse, La Crosse, WI, USA
[2]Department of Computer Science & Software Engineering, Concordia University, Montreal, QC, Canada
[3]School of Information Management, Wuhan University, Wuhan, Hubei, China

Abstract – *With ubiquitous computing, users access their applications in a wide variety of environments. To cope with various and dynamic execution environments, the adaptive mobile user interface is desired to enhance human-computer interactions. This paper discusses the design of the context sensitive mobile user interface that will enable automatic adaptions to the environment. The challenges of this research lie in the areas of context classification and collection, context analysis and mobile user interface adaption due to environmental changes. We propose a design to address these issues and use an e-commerce application to illustrate the ideas presented.*

Keywords: Context, Mobile, User Interface, E-commerce

1 Introduction

Context awareness is increasingly gaining applicability in interactive ubiquitous mobile computing systems. According to Dey's definition [1], "a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". One issue for context-aware applications is to easily make use of these various services at a low development cost and with easy reconfiguration enablers. To address this issue, our research work discusses the collection and classification of the context information, and the automatic adaption based on the context analysis in the mobile user interface behaviors.

A dynamic user environment, which must respond to fast-changing contexts, can benefit from the use of a context-aware device. The e-commerce system is an example of such an environment. Currently mobile computing provides a convenient service for e-commerce systems. It ensures the user's universal access to the system.

Some usability challenges appear in real world e-commerce applications related to entering and retrieving item and user information: 1) human-computer interfaces unsuited for certain highly disruptive user environment; and 2) cognitive excess resulting from the number of steps required to retrieve correct information. We aim to address the idea of classifying the e-commerce system context as a foundation for designing and developing a context-sensitive mobile e-commerce application. The designer must incorporate context-based modifications into the appearance or the behavior of the interface, either at the design time or at run time.

It is important to point out we are separating how context is acquired from how it is used, by adapting mobile user interface features to the user's context. For example: user's technical skills or experience with a mobile device is one of the components in the user's characteristic. The user, as a composite entity, is part of the context. The mobile user interface is automatically adapted to the context information. We hope that this research work will improve human-computer interaction with the aid of a mobile user interface that responds appropriately to contextual changes.

The paper is organized as follows: In section 2, we introduce the application context, which is the e-commerce domain. Note that we do not address the important issues of security, privacy and reliability with regards to e-commerce applications in this paper. In section 3, we present our design approach. In section 4, we compare how our views are similar to those of others and how they are difference. Section 5 concludes the paper and outlines the directions of our ongoing research.

2 E-Commerce Example

In this section we present an illustrative scenario with a mobile context-aware e-commerce application.

Zoe is a client of a famous e-commerce merchant whom often makes many different purchases. Zoe's client profile is used to provide her a customized service. Zoe decides to go by train to visit her friend whom lives near the Grand Canyon. Once the train starts, she turns on her cell phone and uses its Internet connection (e.g. 3G) to connect to the e-commerce server. When connected, Zoe receives "recommendation" on: 1) hiking shoes because Zoe's hobby is hiking. 2) DVDs because today is Zoe's best friend Maddie's birthday, and Maddie's hobby is watching movies. 3) a jacket because the outside temperature measured by the weather station near her current mobile phone location is 25 degrees Fahrenheit.

Just when Zoe decided to look in detail at the products using the application product description, the train enters a tunnel. Inside the train, it gets dark. The mobile device will change to either display the result by sound or display the text in larger font. In addition, Zoe has also configured her profile to download videos only if the mobile phone is using Wi-Fi signals. Thus the application only allows images displayed at the current network environment.

This example shows that the e-commerce ubiquitous application needs to be context-aware in order to cope with different user profiles and preferences, and different elements of the environment in a distributed setting.

3 The Development of Context Sensitive Mobile User Interface

Our approach to adaption is to change the relevant mobile user interface parameters base on context. The context information comes from various sensors built in the mobile device and from the user's profile. The user's characteristic is part of the context in nature. The context information is typically classified into logic and physical categories. User's characteristic belongs to the logic category, while the time and location fall into the physical category.

3.1 Mobile User Interface Features

Our approach allows users to easily handle information on the mobile user interface which deals with the user's motion, and various environmental effects, such as different combinations of ambient conditions (i.e., light and noise levels). The user interface features, which can be changed on a modern smartphone, are listed in Table 1 [2].

Table 1 Mobile User Interface Features

Mobile User Interface Features	Values
Font size	Small, medium, large
Font color	RGB color, black & white
Font format	Times New Roman, Tahoma, etc.
Background color	Auto adjust, changing manually
Data Entry	Typing, tapping, voice
Display information	Text, sound
Message delivery	Text, voice, alert, silent, pre answer
Brightness level	Increase/decrease
Ring volume	Low, medium, high, alert, vibration
Sound level	Mute, regular, loud

Adapting mobile user interfaces to the various basic contexts involved, such as location, time and ambient conditions, will enhance the user's experience of a context-aware device.

3.2 Define User's Characteristics

We considered two general sources for charactering user: domain experience and experience using mobile devices. Our research goal is to adapt the mobile user interface to individual users, while at the same time assigning each user to one of a number of groups. The user's characteristics helped us achieve this goal successfully. For this research, we needed to consider two aspects of modeling: categorization, and differentiation. Through categorization, the differences between people are simplified, and the individuals are assigned to membership groups. Through differentiation, the differences between groups are enlarged, and the differences between individual members of the same group are minimized [3].

In an e-commerce application, the user's domain experience is defined in terms of **VIP** (has been active for more than two years or paid the premium membership) and **client** (has been active for less than two years and did not pay the premium membership). Here *active* means the total value that the user has purchased is over a certain threshold. The mobile experience is defined in terms of **Basic** (less than 1 year), **Intermediate** (more than 1 year and less than 2 years), and **Advanced** (more than 2 years). Example: if a user is in the client group, and his/her mobile experience is basic, then "tapping" is enabled as a default data entry feature in the mobile user interface.

3.3 Rule-based Approach

Our work depends on the internal sensors of a mobile device, user profile and the adaption of mobile user interface features for both entering and accessing data. The key point of the approach is to capture and represent the knowledge required for the mobile user interface to self-adapt at run time or to implement the adaptions at design time. The rule-based approach representation is what we proposed.

A user interface is the link between the software system and the human user, and the software is a tool that helps a user perform his/her task. Rules at different stages can be developed independent of each other. At each stage, different contextual information is included. Example: Zoe is a VIP of the e-commerce application. She wants to 1) check the latest recommendations, and 2) order a DVD for her friend Maddie in a dark and noisy environment. Then the mobile user interface can display the result in large font text, increase the brightness level of the screen; accept the order given by Zoe by tapping, rather than typed on the keyboard, or by voice.

In this scenario, the user interface features adapted or modified (temporarily) are: change to large text output while the environment is dark and noisy; change to tapping since Zoe used all the information that is stored in her profile to place the order. Due to her VIP status, a 10% discount and free shipping are applied to her purchases automatically.

All the condition attributes presented in this work are based on basic context, which consists of the location, time, ambient conditions and the noise level. Domain-based context consists of the user and the tasks. The tasks that a VIP most frequently perform can be summarized in two categories, input tasks and output tasks, as follows:

- Output tasks: review recommendations, watch product video and animations.

- Input tasks: select product, place order, and enter purchase information.

Some sample conditions in the context-aware e-commerce application are listed below:

C1. The mobile phone is in a Wi-Fi environment
C2. The level of light in the room is bright
C3. The level of noise in the room is low

The rules will be based on the match context value. For example,

For a VIP, if [C1] is satisfied, then the mobile user interface features will follow rule 1 as the action A1 and A2 described. It is shown in Table 2.

Table 2 Sample Rule Table for Output Tasks (VIP)

Conditions		**Rules**	
		1	2
C1	The mobile phone is in a Wi-Fi environment	Y	N
C2	The level of light in the room is bright	Y	N
C3	The level of noise in the room is low	N	N
Actions			
A1	Adjust the font size for displaying information to "user default", playing product video and animations	X	
A2	Adjust the display brightness to "user default"	X	
A3	Adjust the display brightness to "high"		X
A4	Receive information by sound		
A5	Receive information by tapping	X	X

Similarly Table 3 shows the sample rule table for input task when the user is a VIP.

Due to the time limit with the current project, we are still in the stage of developing the prototype of the context-based e-commerce application. Table 2 and table 3 are only a sample of the conditions and hence rules. The complete conditions and rules need to be explored and verified through analysis and development. Once we get all the conditions, the complete set of rules is 2^n combinations, where *n* is the number of conditions. Rule analysis later on will be focused on consistency and applicability of the proposed rules.

For the same reason, we have not yet been able to research on the impact of the user's mobile experience in the development of the adaption rules for the mobile user interfaces. For example, an advanced VIP user may desire a more complicated, but efficient mobile user interface.

Table 3 Sample Rule Table for Input Tasks (VIP)

Conditions		**Rules**	
		1	2
C1	The mobile phone is in a Wi-Fi environment	Y	N
C2	The level of light in the room is bright	Y	Y
C3	The level of noise in the room is low	Y	N
Actions			
A1	Adjust the font size for displaying information to "user default", playing product video and animations	X	
A2	Adjust the display brightness to "user default"	X	X
A3	Adjust the display brightness to "high"		
A4	Receive information by voice	X	
A5	Receive information by tapping		X

In summary, our proposed rule-based approach is described in the flow chart below.

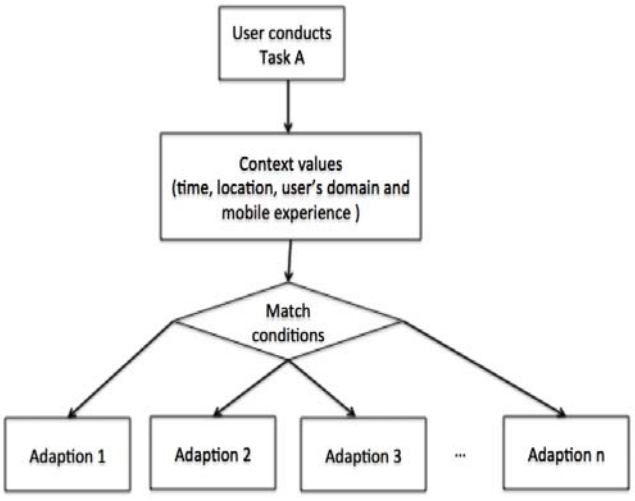


Figure 1 Approach Flow Chart

4 Related Work

Some researchers define context as the user's physical, social, emotional or informational state, or as the subset of physical and conceptual states of interest to a particular entity [4]. The authors in [4] have presented the definition or interpretation of the term by various researchers, including Schilit and Theimer [5], Brown *et al.* [6], Ryan *et al.* [7], Dey [8], Franklin & Flaschbart [9], Ward *et al.* [10], Rodden *et al.* [11], Hull *et al.* [12], and Pascoe [13]. In Dey and Abowd [4], the authors are interested in context-aware systems, and so they focused on characterizing the term itself. In Pascoe [13], the author's interest is wearable computers, so his view of context is based on environmental parameters as perceived by the senses. Our work depends on the internal sensors of a mobile device, and the adaption of the mobile user interface features for both entering and accessing data. Our model is based on separating how context is acquired from how it is used, by adapting the mobile user interface features to the user's context.

Most of the research in this area has been based on analyzing context-aware computing that uses sensing and situational information to automate services, such as location, time, identity and action. More detailed adaption has been generally ignored. For example, input data based on context. In our research, we attempted to build the user's characteristics from both domain experience and mobile technology experience, and to collect all the context values corresponding to the user's task and then to automatically adapt the mobile user interfaces to the context information.

The process of developing context-based user interface has been explored in a number of other projects. Clercks *et al*. [14], for example, discuss various tools to support the model-based approach. Many studies have been conducted on adaption using a decision table. In [15], an approach is proposed for modeling adaptive 3D navigation in a virtual environment. In order to adapt to different types of users, they designed a system of four templates corresponding to four different types of users. Our work differs in that our adaption technique is based on composite context information that extracts values from sensors in smartphones and relates with the user's domain and mobile technology experiences. Then we develop a set of rules for the mobile user interface adaption.

5 Conclusions

Each context-aware application has its own set of behaviors to react to context modifications. Hence, every software engineer needs to clearly understand the goal of the development and categorize the context in the application. We have proposed a rule-based approach and illustrate the idea in an e-commerce application.

The contributions of this research work lie in 1) considering both the user's domain and mobile technology experience in context, 2) detailed modeling inclusion on both input and output data, 3) using the rule to present acquired knowledge in the application. The adaption built into a mobile user interface can enhance the accessibility in the e-commerce domain. The additional benefits are a) increase usability. For example, if the mobile user interface only supports one interaction model, such as typing or voice input/sound output, the usability of the service would be drastically decreased. b) increased awareness of social ethics, e.g. in a quiet room after midnight, the sound could be turned off automatically. c) improved workflow productivity.

The future work of this research will fall into three directions: 1) researching on how the user's mobile technology experience will impact the adaption of the mobile user interface. 2) discovering and verifying the completeness of the conditions and rules. 3) conducting effective testing for the context-aware applications. 4) building a context model and reconfiguring the model for other applications.

6 References

[1] Dey A. "Providing Architectural Support for Building Context-Aware Applications", Ph.D. thesis, College of Computing, Georgia Institute of Technology, Dec. 2000.

[2] Reem Alnanih, Olga Ormandjieva, T. Radhakrishnan, "Context-based and Rule-based Adaption of Mobile User Interfaces in mHealth", The 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH), Procedia Computer Science 21 (2013) 390 – 397.

[3] Oaks P J, Haslam S A, Turner J C, "Stereotyping and Social Reality", Cambridge, MA: Blackwell publishers, 1994.

[4] Dey AK, Abowd GD, "Towards a better understanding of context and context awareness", New York: ACM Press, 2000.

[5] Schilit B, Theimer M., "Disseminating active map information to mobile hosts", IEEE Network 1994, 8(5):22-32.

[6] Brown PJ, Bovey JD, Chen X, "Context-aware applications: from laboratory to the marketplace", IEEE Personal Communications 1997, 4(5):58-64.

[7] Ryan N, Pascoe J, Morse D, "Enhanced reality fieldwork: the context-aware archaeological assistant", In Gaffney, Leusen, Exxon(Eds) Computer Applications in Archaeology, 1997.

[8] Dey AK, "Context aware computing: the cyberdesk project", Technical Report SS-98-02, 1998, 51-54.

[9] Franklin D, Flaschbart J., "All gadget and no representation makes jack a dull environment", Technical Report SS-98-02. 1998, 155-160.

[10] Ward A, Jones A, Hopper A, "A new location technique for the active office", IEEE Personal Commun. 1997, 4(5):42-47.

[11] Rodden T, Cheverst K, Davies K, Dix A, "Exploiting context in HCI design for mobile systems", Workshop on Human Computer Interaction with Mobile Devices 1998.

[12] Hull R, Neaves P, Bedford-Roberts J, "Towards situated computing", 1st International Symposium on Wearable Computers, 1997, 146-153.

[13] Pascoe J, "Adding generic contextual capabilities to wearable computers", 2nd International Symposium on Wearable Computers, 1998, 92-99.

[14] Clerckx, T., Winters, F., and Coninx, K., "Tool Support for designing context-sensitive user interface using a model-based approach", Proceedings of the 4th International Workshop on Task Models and Diagrams, 2005, Gdansk, Poland.

[15] Shi-wei, C. and Shou-Qian, S, "Adaptive 3D navigation user interface design based on rough sets", IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design, 2009, 1935-1940.

cnetmon: Ncurses-based Network Interface Activity Monitor

Steve Hutchinson[1], John Wittkamper[1], Jovina Allen[1], Robert F. Erbacher[2]
[1]ICF International for US Army Research Laboratory, Adelphi, MD 20783
[2]US Army Research Laboratory, Adelphi, MD 20783

Abstract - *This report illustrates the development and use of a network interface activity monitoring tool named cnetmon. This tool is intended to aid system administrators and developers with network-oriented software projects. The main objective for this project was to develop a capability to monitor network activity for all or selected interfaces on a system simultaneously and continuously. We use a display generated by the Linux ncurses library that is updated using a configurable interval. We show added capabilities including interactive response to window-resizing using SIGWINCH. A novel debug-line display capability is provided to show dynamic debug messages on a dedicated line of the display.*

Keywords: network traffic monitoring, network interface, systems administration, ncurses

1 Introduction

cnetmon[1] is a very lightweight command-line tool to display network traffic (packet activity) on any or all of the network interfaces (NIs) on a Linux-based system It uses a ncurses-library-based display that is compatible with any character-based pseudo terminal, and as such, does not require the use of the system graphical user interface (GUI) or Xserver:DISPLAY.

cnetmon is intended for use in the field for remote access into devices such as (network) sensors or other network-attached Linux systems when an administrator with user-level access needs to obtain a dynamic indication of all network traffic entering and leaving that system. Because it does not use the GUI, the complexity and access requirements are very minimal. *cnetmon* can be invoked by any logged-in user, it does not require *sudo* access, and it can operate within a typical secure shell (*ssh*) or *telnet* session.

[1]Throughout this paper, Linux commands are set in an italic font.

2 Motivation

Server farms, cloud computing, compute clusters, and grid computing are all examples of a common technique to combine multiple computer systems into a cooperative network of systems. These systems often intercommunicate using two or more NIs (on each system). Clustered-computers are often rack-mounted for higher density and, as a result, often lack a keyboard or monitor; therefore, they are frequently managed and configured remotely via *ssh* or *telnet* over a network connection. During system configuration, installation, and testing, it is often difficult to determine whether network traffic is being sent and received by each interface.

In general, such systems are built and configured in a central location and then shipped to remote locations to be added to other servers in a system rack or as a single distributed sensor. *cnetmon* allows the installer to observe network traffic from each or all NIs to verify that the system seems properly configured for the installed environment. It also does not require the use of the system GUI or Xserver/client because *cnetmon* will create tabular displays of all traffic using the LIBCURSES library for display on any attached ASCII terminal emulator. *cnetmon* can be used from a remote location, accessed and invoked typically from a *ssh* command-line, and can be invoked by any logged-in user; it does not require root-level access. Many techniques to observe or sample traffic from any NI require super-user privileges, but obtaining elevated privileges is often forbidden, hence a benefit of *cnetmon*.

In this paper, we describe a few use-cases for *cnetmon*. First, *cnetmon* can be used on a laptop computer, which often has two NIs: wired (eth1) and wireless (wlan), along with the internal loopback interface. Laptop-users often must transition between networks without rebooting. *cnetmon* is easily invoked from a command window and will show all NI activities to verify communications to the desired network(s). Second, on a desktop or small server with multiple wired or wireless interfaces, *cnetmon* can show all network activity for each interface dynamically in this more complex network topology. Third, compute-server administration and configuration tasks are often performed using a separate administrative system and command-line tools. *cnetmon* facilitates server configuration and testing and was developed for use in these more complex, multi-network

environments. We frequently use one *cnetmon* window per server during configuration, development, and testing, to obtain a real-time picture of network inter-communications and to verify proper configuration and operation.

3 Related work: bmon

In the search for a user-level, multi-NI monitor, we noticed the "*bmon*" tool [1], which provides indications of network bandwidth utilization from multiple interfaces using the /proc/ file-system [2] and a curses-interface. We use this strategy to implement a curses-based multi-interface activity tool, *cnetmon*, providing various command-line and key-press event-driven parameters to control the display and monitoring update interval.

Although *bmon* was intended to show network bandwidth utilization, we liked its design paradigm using a ncurses display using periodic updates obtained from /proc/net/. Our goal was not to show estimated bandwidth utilization, but to show concurrent network activity measured in terms of packet counts and transfer rates per sampling interval and accumulated for the session.

4 How it works

A long-standing problem for understanding network activity between (Linux or *nix) systems has been the requirement to obtain root or super-user privileges to access and configure devices, such as a NI. *ifconfig* is the Unix or Linux command to display the status of NI devices on a system. Upon executing the *ifconfig* command, the following information is produced on the console, shown below in Figure 1. The first 6 lines pertain to the hardware and network address parameters for each interface as well as the status of the interface. The remaining lines show counts of transmitted and received packets, error counts, and finally the interrupt number and buffers memory location.

```
user@asc2:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:24:81:1c:fd:7d
          inet addr:10.0.0.16  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2601:a:4680:3e6:5cf:ea3d:eed0:64e0/64
Scope:Global
          inet6 addr: fe80::224:81ff:fe1c:fd7d/64 Scope:Link
          inet6 addr: 2601:a:4680:3e6:224:81ff:fe1c:fd7d/64
Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:370 errors:0 dropped:0 overruns:0 frame:0
          TX packets:120 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:46300 (46.3 KB)  TX bytes:20936 (20.9 KB)
          Interrupt:19 Memory:f0500000-f0520000
```

Figure 1. Typical *ifconfig* output.

Although it is true that we could issue *ifconfig* repeatedly to obtain the configuration and counts for network devices, this function call is not intended for repeated invocation to determine network traffic rates. Modern Linux systems provide a /proc/ file system to allow user-level processes to easily read a wide variety of counts for devices; these values are maintained and updated by the kernel in a virtual file system, /proc/. The /proc/ file system was originally intended as a way to provide information about processes in a system. As such, it also was a convenient means of exposing kernel information to a structured file system requiring only user-access rights to read this information. A corresponding application programming interface (API) is provided for read and write access — using sysctl (system control) calls to configure parameters of the running kernel [3]. This capability was gradually introduced into Unix systems starting as early as 1984; the current implementation in Linux is as an extended, virtual file system contained only in memory and has directories for other kernel information categories such as kernel-modules, file-systems, interrupts, and devices including NIs, kernel messages, drivers, and CPUs.

The *cnetmon* executable periodically examines the /proc/net/dev file on the Linux system. These values are sampled on each loop cycle (by default, one second), which is configurable on invocation or by pressing a number-key while running. Linux systems also maintain an uptime value, the number of seconds since last rebooting. *cnetmon* saves this date-time value at launch (fork) time and displays the session length time in the screen header section

Contents of /proc/net/dev:

```
Interface
lo:
   bytes          570671
   packets        6267
   errs           0
   drop           0
   fifo           0
   frame          0
   compressed     0
   multicast      0
   bytes          570671
   packets        6267
   errs           0
   drop           0
   fifo           0
   colls          0
   carrier        0
   compressed     0

eth0:
   bytes          14797900909
   packets        17797994
   errs           0
   drop           0
   fifo           0
   frame          0
```

compressed	0
multicast	3120
bytes	4116686178
packets	14414011
errs	0
drop	0
fifo	0
colls	0
carrier	0
compressed	0

5 Implementation

The design goals and requirements for *cnetmon* are to periodically examine the network device-file in the /proc directory on a Linux system to:

- Enumerate NIs
- Collect traffic statistics
- Convert traffic counts to display quantities and units
- Allow a variety of command-line arguments

We also provide a release make/build capability for most Linux systems (including embedded devices, such as Raspberry Pi, etc.)

After initialization during which command-line arguments are parsed, *cnetmon* enters the main_loop. With each pass through main_loop, it obtains new counts for packets, bytes, errors, drops, collisions, etc., and calculates display values as requested updating the ncurses display at the end of each interval. Display values are calculated from the following:

Li	update loop interval, in seconds
Tu	Linux uptime in seconds (since reboot)
Tnow	current Linux system time, epoch time seconds
T0	*cnetmon* invocation start timestamp in epoch time seconds
Ls	session time length in seconds: (Tnow – T0)
P[i]	packet count parameter from /proc/net/dev, at time interval = i
B[i]	byte count parameter from /proc/net/dev, at time interval = i

For each interface and at each interval:

$$SessionPKT\ s = P[Tnow] - P[T0] \quad (1)$$

$$IntervalPK\ Ts = P[Tnow] - P[now - i] \quad (2)$$

$$SessionRat\ e = (B[Tnow] - B[T0])/1000 * Li] \quad (3)$$

$$IntervalRa\ te = P[Tnow] - P[T0] \quad (4)$$

Command-line programs used for monitoring often generate display data output in the form of one-line records and then render them into a scrolling console window. Very wide, or multi-line records, when scrolled like this, are difficult to understand. Since network interface data is of this nature, a scrolling display will be difficult to use. Instead, we use a display technique that renders these parameters in strict rows and columns such that the location of each on the screen does not change. This tabular process makes the changing parameters more obvious. Cell contents can change with the fixed regularity of the chosen update loop interval. Although this is a somewhat primitive display technique compared with GUI implementations, such a capability is easily provided by the Linux, Ncurses library. Ncurses allows development of rather sophisticated tabular displays, useful in situations in which a GUI display is unavailable (as would be the case for many “headless” server or compute-clustered environments).

6 Ncurses library

Ncurses [4] stands-for “new” curses—a reimplementation of the “curses” library to use a text-based terminal to emulate a more dynamic interface that has some attributes of a modern GUI. Curses was originally developed at the University of California at Berkeley for a Berkeley Software Division (BSD) release around 1980. Ncurses contains enhancements to curses and was made available starting in the mid-1990s under a “Permissive free software license” and not the General Public License (GPL) to afford wide redistribution and linking to this library.

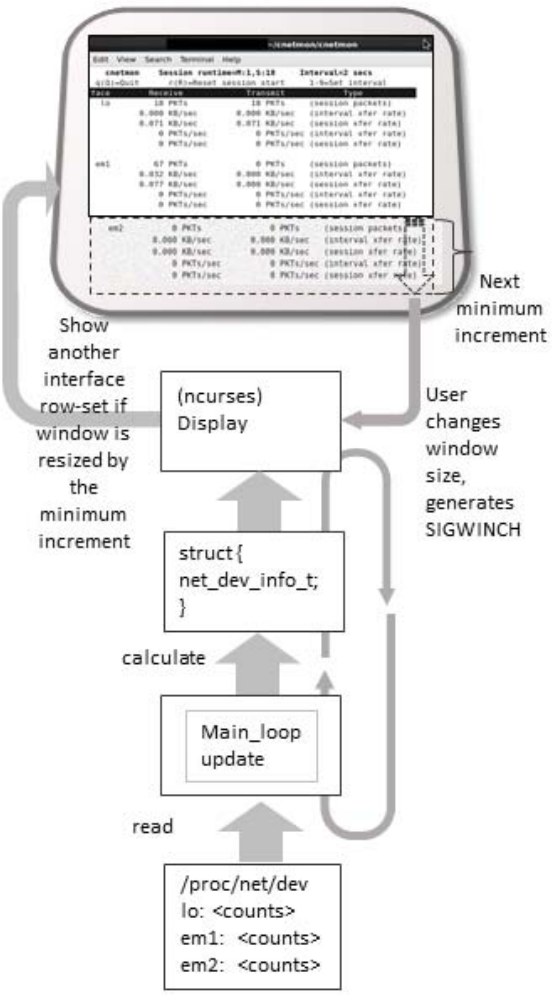


Figure 2. Resize of window to reveal additional interface row-sets.

cnetmon, like many other ncurses programs, obtains terminal window geometry parameters from the terminal emulator when the program is launched. The combination of command-line switches will determine the number of rows needed to describe each interface; by default, the display will require one row per interface with the addition of three header rows. Use of the "-a" switch will result in the display of 7 or more rows per interface. *cnetmon* calculates how much space (height) is needed, and then it only displays as many interface row-sets as can fit in the current window geometry.

Use of up/down (U/D keys) allows the user to scroll up or down an interface (row-set) at any time. To avoid requiring the user to quit, resize the terminal, and re-launch in order to see additional interfaces, we support dynamic changes in window size using the SIGWINCH signal (window change), which is supported by most terminal emulators. When the user changes the window geometry, the program receives the SIGWINCH signal and obtains new window geometry. *cnetmon* recalculates the number of interface row-sets that will fit in the new window and updates the display generation parameters in ncurses without resetting any of the current packet counts and rates.

Figure 2 illustrates the various sub functions within main_loop, showing the generation and response to window size changes. Figure 3 below shows the help message with option switches and their meaning.

Help message: cnetmon -H

cnetmon [aD:ehHi:Lm:n:rtTu:]

-a	Show errors, data rate & totals (-ert)
-D #	Debug level (0-15)
-e	Show error data
-H	Help message
-i name	Ignore interface "name"
-L	List interfaces (with some statistics)
-m name	Show only interface "name"
-n #	Show total bytes for system uptime
-r	Show data rate
-t	Show data totals
-T	Show total bytes the "Quick" display
-u #	Update frequency, seconds (default 1)
Interactive:	
d/D	Scroll down interface list
q/Q	Quit
r/R	Reset Session time
u/U	Scroll up interface list
1-9	Load value into interval time

Figure 3. Usage help message.

7 Usage scenario

An actual usage scenario is shown below. We have an existing Linux server (Ubuntu 14.04 server) that will be used to provide various services to three separate networks, shown in Figure 4 as Internet, MeshNet_1, and MeshNet_2. This server does not have an attached display. We use a 2nd system with a terminal emulator and establish *ssh* session to the server. We copy the *cnetmon* executable onto our /home/user/ directory using *scp* (secure copy command). This session is established through the Internet and gateway attached to 'eth0'. Invoking *cnetmon*, we easily observe network activity on eth0 and no activity on eth1 or eth2. *cnetmon* does enumerate other interfaces such as the local loopback (lo) and a virtual bridge for use by associated libraries to offer network address translation (NAT).

It is normal for local loopback to accumulate and show significant traffic during network traffic sessions as it is used for process-process communications. We then connect a second network (MeshNet_1) gateway to eth1. This interface had been configured already to accept a DHCP-issued

address. *cnetmon* clearly showed packets corresponding to DHCP requests and lease responses. After obtaining another user shell (*ssh*) to the server, we were able to access the web admin service on the gateway—to continue configuration of this network.

We then connected MeshNet_2 gateway to eth2. *cnetmon* observed no activity on eth2. This required further investigation. /etc/network/interfaces is the configuration file used by Linux systems to initialize and configure all NIs. eth2 had not yet been configured, and it was activated by adding the following to /etc/network/interfaces (these must be done as admin or root access):

auto eth2

iface eth2 inet dhcp

This change required restarting the networking services:

sudo /etc/init.d/networking restart

cnetmon showed no eth2 activity after restarting the network services. Next, we tried a *shutdown* – reboot which did reconfigure the interfaces and driver. After rebooting, *cnetmon* showed activity on all three physical NIs as well as the virtual loopback interface.

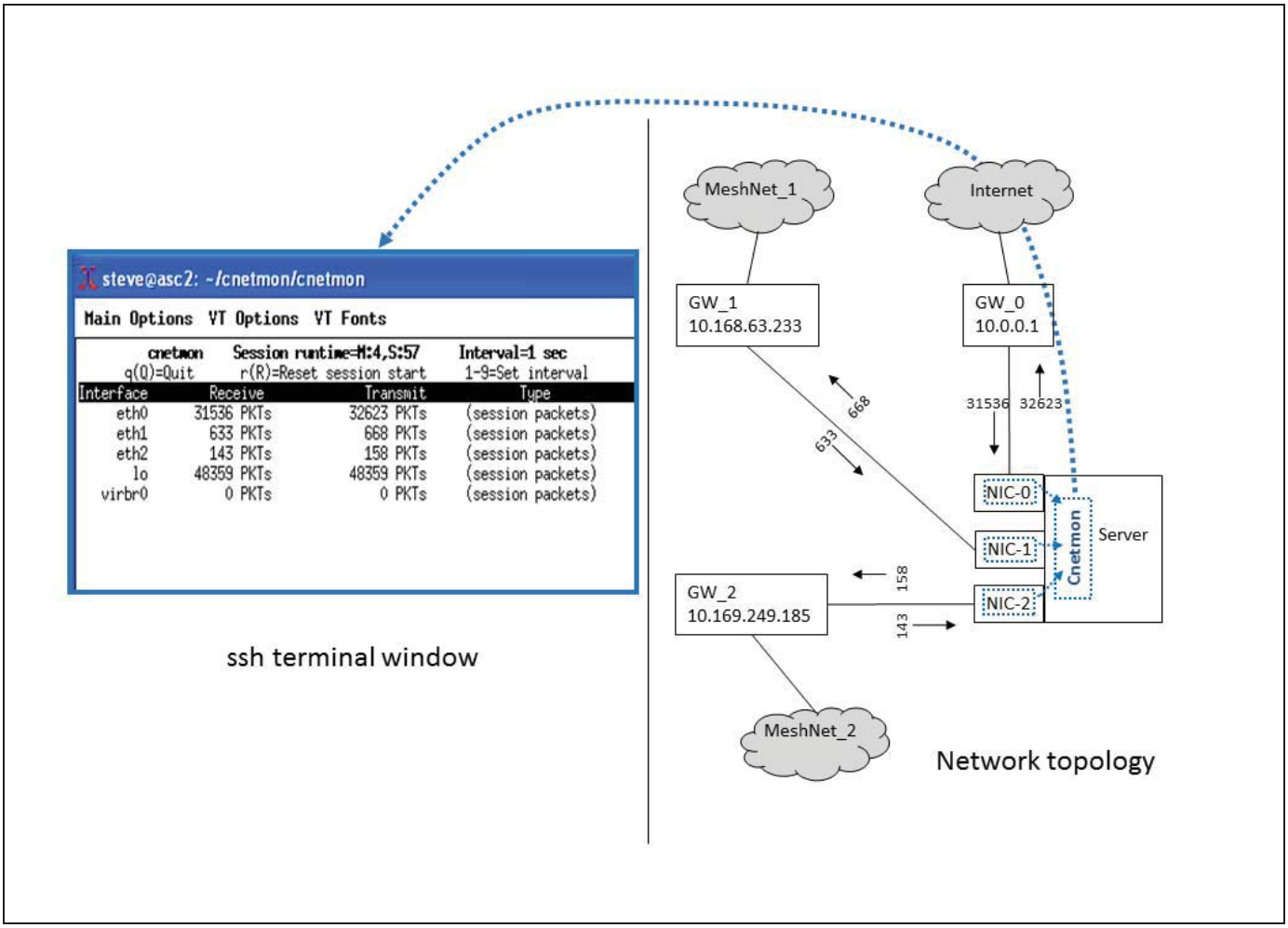


Figure 4. Server with connection to 3 networks.

If the server had been pre-configured prior to installation, it is likely that *cnetmon* would have allowed us to observe and verify each of the network gateway additions in real-time at power-on. In this case, additional configuration requiring root-level access was required. We were able to observe resulting network activity in real time using *cnetmon* in a second session window.

8 Compute-server example

To illustrate additional capabilities of *cnetmon*, we show results from running it on a blade-server with 5 NIs. This type of server is common today and is used to populate the many rack spaces at internet and content hosting facilities. Although this server has 5 NIs as shown in the DEBUG: line, the display window geometry affords

space for only 2 complete record sets, (lo) and (em1) shown in Figure 5. This server has been up (running) for just over 105 days and *cnetmon* has been running for 85 seconds, updating at 1-second intervals.

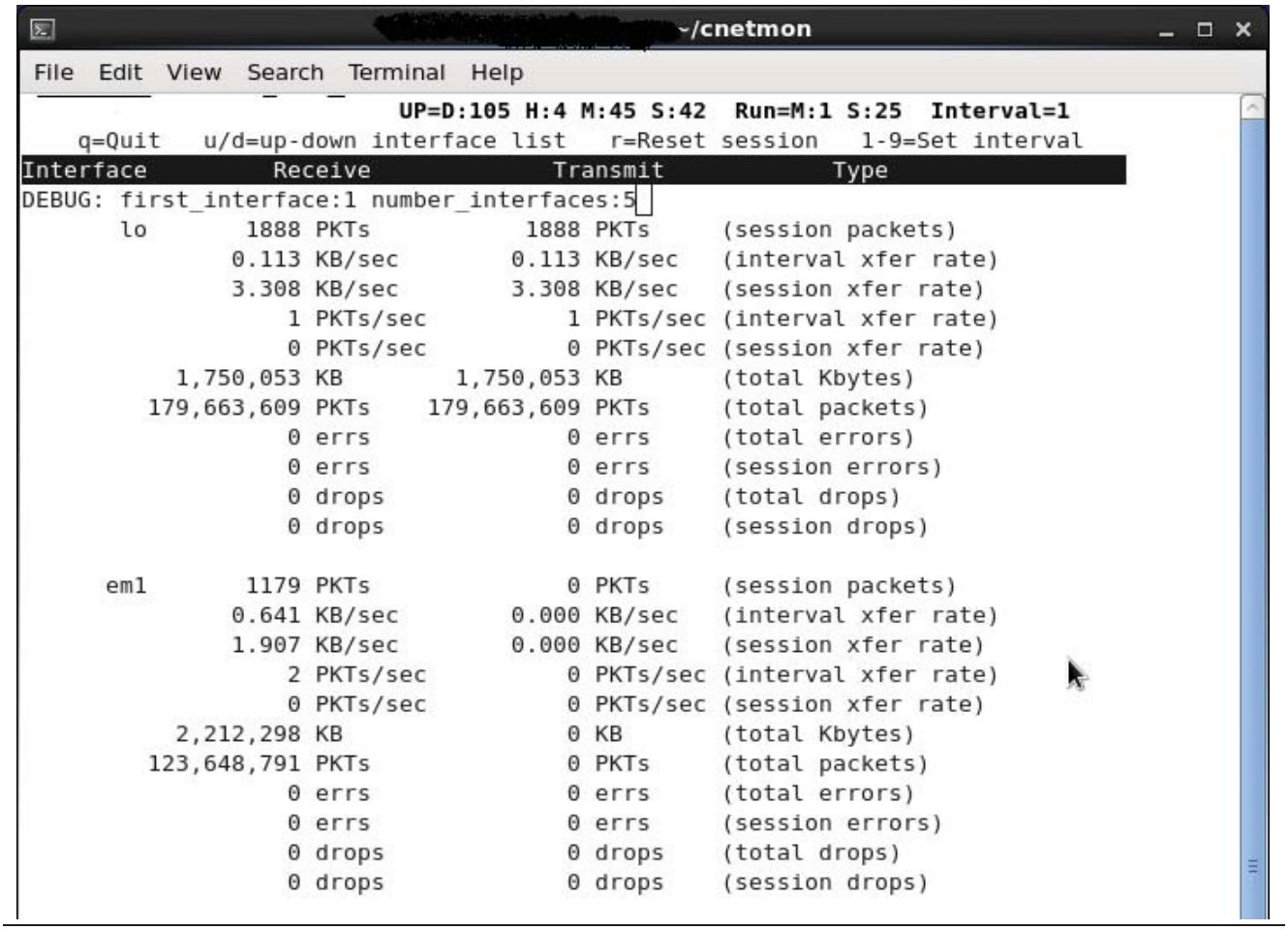


Figure 5. *cnetmon* –D 1 –r showing counts and rates for the session and for the last main_loop interval. Notice this also shows the –D flag, which adds an additional debug-message line to the display. Here, a cd_printf statement has been included to show the first and total number_interfaces available.

9 Debug print

Coding and debugging an ncurses program can be very challenging. To facilitate debugging, we incorporate a debug display activated using a command-line switch. The code snippet below from cnetmon.c :main() illustrates how to print messages to the debug message line using the '-D 1' command-line argument.

```
// Step through device list
for (j = 0, i = first_interface; i <= number_interfaces; i++)
{
  int display = 1;
  if (  !prog_flags.first_time || prog_flags.match_inface )
  {
    display = 0;
  }
}
////////////////////////////////////////
cd_printf("first_interface:%d number_interfaces:%d",
first_interface, number_interfaces);
////////////////////////////////////////
refresh ();     // Update display
```

10 Conclusions

We have shown how *cnetmon* can provide easy access to the network activity from multiple interfaces, on multiple systems; however, the executable must first be available on each system. Therefore, we intend to provide *cnetmon* to be available as openSource code, providing the sources, documentation, a makefile, and a pre-compiled, 32-bit binary. Although most systems today are 64-bit architecture, the precompiled 32-bit binary should run on almost any Linux operating system. A sophisticated developer-user can re-compile *cnetmon* from the sources, possibly adding new features and debugging cd_printf statements to facilitate the application and intended uses.

We also will approach major Linux packagers and distribution groups, notably Red Hat, Fedora and Ubuntu, to encourage inclusion of *cnetmon* in future distribution releases.

11 References

[1] Travis Graf. "bmon – bandwidth monitor and rate estimator", retrieved from https://github.com/tgraf/, June 15, 2014.

[2] Terry Dawson. "Exploring the /proc/net/ directory," O'Reilly, retrieved from http://www.onlamp.com/pub/a/linux/2000/11/16/LinuxAdmin.html, March 26, 2015.

[3] M. Tim Jones. "Access the Linux kernel using the /proc filesystem", IBM developerWorks Technical Library, 2006, retrieved from http://www.ibm.com/developerworks/library/l-proc/index.html, April 15, 2015.

[4] Free Software Foundation. "Announcing ncurses release 5.9". Free Software Foundation, 2011, retrieved from https://www.gnu.org/software/ncurses/, March 26, 2015.

Political Risk Mitigation Model for Software Development Budget

Esiefarienrhe Michael Bukohwo[1], Wajiga Greg[2]
[1]Department of Computer Science, North-West University, Mafikeng, South Africa,
[2]Department of Computer Science, Modibbo Adama University of Technology, Yola, Nigeria.

Abstract *- Political risk models attempts to evaluate country risk and how they affect investment and repatriation of funds. This paper applied the concept of political risk to software development and discusses how such risk could affect software development project by increasing software development budget. It further developed a model to demonstrate the practical application of political risk to estimating political risk probabilities and their effects on project budget. The crises scenario in the Niger-Delta of Nigeria provided a platform for model simulation. First, a vital input needed for the model was project development cost. This was obtained from an on-going software development project. Next, various political risk were identified and the probabilities of affecting software cost elements were estimated with regard to cost elements. The model was used to computes the adjusted budget for the software development project that mitigated the identified political risks. The results obtained shows that political risk can cause software project failure, introduce bugs and invariably affects the project completion schedules. It was shown that if the model can be applied to software development, there will be less software failures and its consequences on the economy.*

.

Keywords: Software Risk, Political Risk, Model, Software Development, Adjusted budget

1 Introduction

Software risk management is a broad area in software engineering that requires researches in order to fully harness the various advances in software development methodologies, languages and tools. Developing safe and efficient software requires that all risk components inherent in the processes involved in software product development must be fully analysed, resolved and mitigated. This reinforces users confident in computer application to problems solution and paves ways for institutionalizing and globalizing information technology.

Software has become paramount to our everyday activities. Why is software so important? Software flies our airplanes [1], controls our automated teller machines (ATM) [2], and even controls our car engines [3]. The implications are that software drives any modern economy. The extent of the application of software to the economy varies with countries. The efficiency of these applications also varies. A fact that is easily established is that organizations are more eager to acquire information technology applications without recourse to risk implication. This has become a fundamental problem if not addressed scientifically will lead to serious consequences and may even endanger lives and properties.

Large software projects will never be without some risks but if risks can be brought down to acceptable levels, that will be a good beginning [4].

Everyone has an intuitive understanding of risk but how can understanding risk help software to be more successful? First, we need to understand that a risk is not a problem. Rather, a risk is something that might occur in the future: a possibility, not a certainty. To be technically precise, there are two factors that comprise a risk:

1. Probability or likelihood that it will occur.
2. Loss resulting from its occurrence.

The term "risk" has been erroneously used as a synonym of "uncertainty" and "threat" [5, 6, 7]. Risk in software is viewed as a measure of the likelihood of unsatisfactory outcome and a loss affecting the software from various perspectives: project, process and product [5,7]. However, this definition of risk is misleading because it confounds the concepts of risk and uncertainty. According to [8] most part of decision-making in software processes are under uncertainty rather than risk. Uncertainty is a situation in which the probability distribution for the possible outcome is not known.

[8] therefore defined risk as the product of the value of an outcome times its probability of occurrence. While risk indicates a probabilistic outcome; threat is used to identify the danger that can occur.

[9] see r**isk** as a function of the **likelihood** of a given **threat-source's** exercising a particular potential **vulnerability**, and the resulting **impact** of that adverse event on the organization.

From the above definitions, risk can be looked at from a number of different perspectives. First, risk concerns future happenings. Second, risk involves change. The third aspect of risk involves choice.

Authors in [10] defined software development risk as the exposure to one or more of four types of risk:

i. Performance risk, or the failure to obtain all of the anticipated benefits of the systems and software under development
ii. Cost risk, or significantly exceeding budgeted or estimated cost
iii. Schedule risk, or the failure to deliver satisfactory software products by scheduled milestones and user need dates
iv. Support risk, or the delivery of a product that has excessive lifecycle maintenance costs due to deficiencies in maintainability, flexibility, compatibility or reliability

Software Risk Management is a set of practices that enable software development projects to assess overall project risk and identify, prioritize, and manage specific risks. While [11] defined software risk management as the practice of controlling risk that have the potential for causing unwanted program effects. This control is an entire development life cycle activity, starting with planning for risk at the earliest stages of the project and continuing with monitoring and alleviating risk through the support stages. [12] sees software risk management as a process which seeks to identify, address, and eliminate risk items before they become either threat to successful software operation or major sources of software rework.

2 Analysis of Researches Related to Software Risk

There are presently three main groups of researches related to software risk namely:

1. Assessing Software risk by measuring Reliability of the Project.

This group of researchers follows a probabilistic approach to assess the reliability of the software product [13, 14, 15, 16]. This approach assesses the reliability of the software product and not the risk inherent in failing to complete the product within constraints. The papers cited above concern themselves mainly with techniques to assess risk related to failures of software projects. The problems with these approaches from a realistic perspective is that the resulting assessments arrive too late to economically correct possible faults, because the software product is mostly complete and development resources are mostly gone at the time reliability of the product can be assessed through testing.

2. Assessing Software Risk by using Heuristic Approach

Some researchers assess the software risk from the beginning of the project, in parallel with the development process. However, these approaches are less rigorous, typically subjective and weakly structured. These approaches uses list of practices and checklist [5, 7, 12, 17] or the use of scoring or grading techniques [6].

3. Assessing software Risk using Macro Model Approach

A third group of researchers uses well known estimation models to assess the risk inherent in software projects. The widely used methods are COCOMO [18], and SLIM [19]. Both assume that software requirements will remain unchanged, and require an estimation of the size of the final product as input for the models [20]. The size of the software product cannot be actually measured until late in the project.

2.1 Political Risk Assessment

In his examination of political risk as used by North Atlantic multinationals, [21] came to the conclusion that a corporation's strategic planning is an important determinant of its profitability and that environment scanning, including political risk assessment is a vital input to the process of strategic planning.

The approach to political risk assessment can be classified into two namely the subjective and the objective approach. The objective approach attaches importance to methodological and procedural solutions to the assessment of political risk. Proponents of objective approaches therefore view the method and procedure as bulwarks against the fallibility and limitations of human judgement. The objective approach could also be referred to as formal-oriented approach. The subjective approach makes use of human judgement, intuition and experience to predict and forecast the evolution of the political environment. While method-oriented approaches work on within the context of statistical data and models, the subjective method make intensive use of survey, advice and judgement from specialists and consultants.

2.1.1 Subjective Approaches

The classical subjective approaches to the assessment of political risk includes three methods; the Grand Tour, the old Hands and the Delphi techniques.

Grand Tours

The technique uses impressions and information are gathered through some preliminary market research or an inspection tour. These impressions and information can be gained through contact with local leaders, government officials and businessmen or through survey of the political landscape. All the information and impressions gathered through this short survey of the political landscape are then analyzed and evaluated [22].

Shortcomings of the Grand tour approaches reside in its vague nature and overdose of selective information.

Old Hands

Through the old hands methods, multinationals seek to acquire area or country expertise from diplomats, journalists, businessman or firm experts on a consulting basis. The

expertise of diplomats, journalists or business usually includes assessment of the objectives and personalities of a country's current leadership, the strengths and weaknesses of competing political groups and the likelihood of new legislation [23]. The drawback of this approach is its unsystematic character and the fact that it is based on the judgments of outsiders.

Delphi techniques

The Delphi techniques offer an example of a more elaborate and systematic use of human judgment and experience. In the first part of the Delphi technique, corporate decision makers try to identify selective elements which could influence a nation's political destiny: size and composition of the armed forces; delays experienced by the foreign investors; political kidnappings etc. Next, a wide range of experts is asked to rank or weigh the importance of these factors for the country under consideration. Then responses are collected and a checklist of the ranked variables is constructed. Finally, the corporate decision makers aggregate the ranked variables of the checklist into an overall measure or index of political risk [24].

Shortcomings of such an approach involve the possible deficiency of the relevant questions and the fact that a single addition and classification of political variables without taking into consideration other variables such as social, cultural and economic variables is misleading and inaccurate. Summarily, the main shortcomings of subjective approaches are due, to a large extent to the fact that they rely largely on intuition and human judgment.

2.1.2 Objective approaches

One way to overcome the above mentioned shortcomings of the subjective approaches consists of making intensive use of quantitative data on political factors and of the econometric and probabilistic methods to improve the accuracy, the precision and the predictability of political events. One important objective econometric technique employed for the objective measurement of political risk is multivariate analysis (MVA), which make multidimensional decisions possible. The MVA could provide a very precious source of information for analyzing complex issues such as political risks. The MVA can be classified on the basis of two possible uses: (1) to predict future political trends on the basis of current and historical information or (2) to describe more fully underlying relationships affecting a nation.

The distinguishing feature of predictive techniques is that one or several variables are said to be a function of some other variables. Multiple regression is in fact one of the predictive techniques used by decision-makers when the data are quantitative or numerical.

One of the shortcomings of the quantitative approaches is related to the inherently complex and subjective nature of the political risk. Many important political issues seem to defy quantification, and decision-makers are often forced to rely on their judgment and intuition to a greater degree than may be desirable. Now, most corporate approaches to political risk assessment stress methodological and procedural solutions to the problem of political risk because of its precision and accuracy. Yet as suggested by [25], all objective methods work on and within the context of a well defined model. The model is treated as the problem and the problem is identifiable to the model. Results drawn from the model are interpreted as conclusions on the problems itself, assuming that the problem structure matches or comes very close to that of the model. Thus political risk is seen as an objective attribute of the problem to be uncovered, measured and quantified through its counterpart in the model. Moreover, the method-oriented approaches imply conceptually that the decision-maker should play a passive role. He should uncover and bring out what is already inherent in the problem, but is not thought of as playing an active role (like in subjective approaches) in bringing structure to the problem and perceiving and defining the nature of political risk within this structure.

There are a number of researches related to political risk from objective approach but they are in the field of economics, investment and finance, [26, 27, 28, 29, 30, 31].

3. Material and Methods

The data used for this research were obtained from both primary and secondary sources. The Primary sources of data includes the use of questionnaires, interviews and observations while the secondary sources include review of existing literatures, system documentation manuals and the review of existing system source listing.

Interviews and personal observations were carried out for the purpose of finding facts about existing methods and to investigate how organisations handle risk when developing software. In most situations, existing documentation about the organisation system were reviewed. This helped in fact-finding about the existing system and problems faced. The use of these fact-finding techniques enables the researcher to understand the methods used to derive solutions to problems.

The Structured System Analysis and Design Methodology (SSADM) which is a Software Engineering Methodology that involves system decomposition to sub-system and the systematic analysis of each sub-system were adopted. Flowcharts and Pseudo codes were drawn and written for the proposed solution.

The Expert System methodology, which involved knowledge engineering process of inference and knowledge-based, is also adopted.

3.1 Political Risk Model

The development of efficient, reliable and maintainable software is the wish of every programmer in an ideal situation. An idea situation may refer to a rational design,

human capacity, technological or other external factors which may be practically outside the control of the developer. One of such external factor that is particularly interesting is the presence of political activities within the software development environment which constitutes various forms of political risk. Political risk is viewed as an external activity layered on the development process. Previously, these external factors are not recognised as having any impact in software development activities. This hitherto, was a wrong assumption. There are evidences to show that software projects have been abandoned or derailed due to war, riots, new government fiscal and monetary policies, government changeover etc. Software development budget have become unstable, schedules slippages, unreliable software due to various political activities impact on the organisation and subsequently affecting the software project. The effects of political risk in software development are varied. Of importance in this research is the effect on the developmental cost of the project. This includes the effect on the mobilization and co-ordinating cost of the team and other procurement cost as well as availability of personnel and motivation. Given that cost is an important factor of production, its inadequacy or non-availability due to inflation, monopolies etc may constitute political risk that may derailed the project.

In order to mitigate these environmental factors particularly as it relates to political risk, the following model is proposed to handle the mitigation of political risk.

Political risk is modeled as a multiplicative function of vulnerability and cost:

$$E_k = \sum_{i=1}^{N} (P_i + 1) + C_k$$

$$PB = \sum_{k=1}^{M} E_k$$

$i = 1 \ldots N$ *Political events*, k = 1…M cost items

$$0 \leq P_i \leq 1, \quad C_k > 0$$

Where

$P_i = prob.of\ a\ given\ event\ i$

$C_k = cost\ of\ item$

$E_k = cost\ of\ a\ given\ item\ under\ political\ risk$

PB = Budget cost of software project under political risk

This model is used to estimate the probabilities of the identified risk occurring in relation to the effect on the budget of a specific item of cost. The proposed solution uses both the subjective and the objective approaches in the estimation of political risk probabilities.

The estimated cost (budget) for the software development is extracted from the project database in Table 1.0.

With the assistance of the organisation's System Analyst, we studied the prevailing political situation and its effect on the project. This was done by estimating the probabilities of each political risk according to the effect it will have on a given element of estimated cost. Although, the probability estimate was a subjective measure, it was however done in conjunction with political analysts who are quite familiar with the political terrain of the country.

3.1.1 System Input

The system requires two vital inputs namely:

i. The budget for the system development and

ii. The list of political events that could constitute risk to the project

Table 1.0 and Table 1.1 show the input to the system while Table 1.2 shows the output from the system in tabulated form.

Table 1.0: Estimated Cost (Budget) for the Development of e-Cataloguing and Artifact System

	₦	US $
Team Mobilization	5000	43
Team Syndicate Grouping	3000	26
Data Collection/Transportation to Site	40,000	342
Stationary	20000	171
Digital Cameras Additional	7000	60
Memories	4000	31
Allowances	70000	598
Case Tools	10000	85
Input	25000	214
Processes	40000	342
Output	10000	85
Databases	30000	256
Hardware Cost	50000	427
Software Cost (Language Compiler)	20000	171
SLOC (N100per line for 1000 lines)	100000	855
Linkages	10000	85
Testing/Debugging Tools	10000	85

* Conversion rate of ₦117 to 1 US$ was used

Table 1.1: Political Risk with Serial Number assigned for identification

Political Risk	S/N
Criminality	1
Putsch	2
Civil War	3
Civil Disobedience	4
Riot	5
Corruption	6
Fragile Political Structure	7
Regulatory Monopolies	8
Blocks to Capital and Profit Repatriation	9
Local Content requirement rule	10
High taxes and weak incentive	11
Looting	12
Destruction of houses and properties	13
Rape	14
Food scarcity	15
Fear and tear	16
Restriction and movement	17

Table 1.2 shows in Tabular form the output generated from the input of Tables 1.0 and 1.1. Columns A and B represent the system development budget. Column C shows the various risk (represented by serial Numbers - see Table 1.1) associated with each cost element (limited to 3 risk per cost element). Column D shows the estimated probabilities of political risk (Column C) occurring (converted to percentages) as it affects cost elements (Columns A and B). The values in Column D are derived by multiplying $C_k P_i$ (e.g. 5000 x 2%, 5000 x 3%, 5000 x 2%). Column F is the summation of Column E using $\sum_{i=1}^{3}(P_i + 1) + C_k, \quad k = 1 \ldots 17$

It is noteworthy to say that Column F gives cost of each cost element adjusted as a result of political risk. It is the summation of these various values that gives the total cost of developing the software under political uncertainty i.e. $\sum_{k=1}^{M} E_k = 477{,}060$

The difference between the former budget (454,000) and the new cost (477,060) shows an injection of additional slack funds of 23,060 to ensure project sustainability.

3.1.2 The System Output

Table 1.2: Output from the System in Tabular form

(A) Project Cost Components	(B) Budgeted Cost (Ci) (₦)	(C) Political risk affecting Project Cost Components	(D) Political Probabilities (Pr) (%)			(E) Probabilities Values			(F) Adjusted Component Cost (₦)
Team Mobilization	5,000	16, 17, 5	2	3	2	100	150	100	5,350
Team Syndicate Group	3,000	3, 4, 16	1	2	2	30	60	60	3,150
Data Coll /Transp. to Site	40,000	3, 4, 5	3	4	3	1200	1600	1200	44,000
Stationary	20,000	3, 5, 17	1	0.5	2	200	100	400	20,700
Digital Cameras	7,000	13, 11, 17	4	3	2	280	210	140	7,630
Additional Memories	4,000	11, 9, 8	3	2	2	120	80	80	4,280
Allowances	70,000	11, 9, 8	1	2	1	700	1400	700	72,800
Case Tools	10,000	15, 17, 16	4	1	2	400	100	200	10,700
Input	25,000	9, 10, 8	1	2	2	250	500	500	26,250
Processes	40,000	11, 8, 12	1	2	1	400	800	400	41,600
Output	10,000	6, 11, 17	2	1	2	200	100	200	10,500
Databases	30,000	17, 11, 6	2	1	1	600	300	300	31,200
Hardware Cost	50,000	11, 17, 6	2	1	2	1000	500	1000	52,500
(Language Compiler)	20,000	11, 10, 6	3	1	3	600	200	600	21,400
SLOC (N100per line for 1000 lines)	100,000	11, 10, 6	1	1	2	1000	1000	2000	104,000
Linkages	10,000	17, 16, 15	2	1	3	200	100	300	10,600
Testing/Debugging Tools	10,000	17, 6, 1	1	2	1	100	200	100	10,400
TOTALS	**454,000**								**477,060**

4 Conclusion

The concept of Political risks was introduced and referred to it as external factors because it is not within the control of the developer when building the product. The developer should be aware of such factors and incorporates from the beginning metrics to mitigate such factors should they arise. The model developed can handles political risk in software development by building financial slacks based on systematic methodology into the development budget so as to ensure project sustainability in the wake of political risk occurrence.

This is necessary as the most vivid effect of political risk results in inflations, fear and panic buying. Project development personnel may be unwilling to continue with the project and there is need to urgently recruit more staff, procure more facilities and pay extra cost for almost everything. From experience of software development projects in times of political tumor, access to additional funds through proposals is almost impossible. This from experience account for most of the problems software developer faced and this is responsible for project failure.

It is noteworthy to say that system analyst must be very careful in the estimation of the various probabilities of political events occurring. We strongly believe that this step should be handled by political scientist in conjunction with the analyst. Also the adjustment coefficient to use depends on the output of the political dependences. This model when used by analyst will result in developing timely, efficient and robust software product(s) irrespective of the political situation.

5 References

[1] Tomayko, J. E. (1991). The Airplane as Computer Peripheral. American Heritage of Inventions & Technology, **7**(3): 56-60.

[2] Kanter, J. Schiffman, S. and Horn, S. (1990). Let the customer do it; from grocery robots to photo kiosks, computerized selfservice. *Computerworld*, 24: 35

[3] Woolnough, R. (1990). TI drives at Euro autos; makes 'PACT' to win microcontroller business. *Electronic Engineering Times*, August-December.

[4] Jones, C. (1994). *Assessment and Control of software Risks*, Yourdon Press, Prentice Hall, Englewood Cliffs, N.J

[5] Higuera, R. and Haimes, Y. (1996). *Software Risk Management. CMU/SEI-96-TR-012*.
IEEE, (Std 1074-1991IEEE); *Standards for Developing Software Life Cycle Processes*

.[6] Karolak, D. (1996). *Software Engineering Management*, IEEE Computer Society Press, Washington DC.

[7] Hall, E. (1997). *Managing Risk, Methods for Software Systems Development*, Addison Wesley, Reading, Mass.

[8] Nogueira, J. (2000). *A Formal Risk Assessment Model for Software Projects*. Ph.D Dissertation. Naval Postgraduate School.

[9] Stoneburner, G. Goguen, A. and Feringa A (2002). Risk Management Guide for Information Technology Systems, NIST Special Publication 800-30.

[10] Software Management Guide Volume II, Software Technology Service Center, Hill AFB, October 1993

[11] Edmund, H. (1997). Managing Risk in Aerospace Programs. *Aerospace America*, **35**(4): 36-39.

[12] Boehm, B. (1991). "Software Risk Management: Principle and Practices", *IEEE Software*, **8**(1): 32-41.

[13] Schneidewind, N. (1975). Analysis of Error Processes in Computer Software, Proceedings of the International Conference on Reliable Software, *IEEE Computer Society*, **21**-23: 337-346

[14] Fairley, R. (1994). Risk Management for Software Projects. *IEEE Software*, **11**(3): 57-67.

[15] Lyu, M. (1995).*Software Reliability Engineering* IEEE Computer Society Press, Washington DC.

[16] Musa, J. (1998). Software Reliability engineering: More Reliable software, Faster Development and Testing, McGraw-Hill.

[17] Charette, A. K and White, M. (1997). Managing Risk in Software Maintenance, *IEEE Software*, **13**(4):110-117.

[18] Boehm, B. (1981). "Software Engineering *Economics*", Prentice-Hall, Englewood Cliffs, N.J.

[19] Putman, L. (1980). *Software Cost and Estimation and Life Cycle Controls: Getting the Software Numbers*. IEEE Computer Society Press,Washington DC.

[20] Londeix, B. (1987). *Cost Estimation for Software Development*, Addison-Wesley, Reading, Mass.

[21] Stapenhurst, F. (1992). Political Risk Analysis around the North Atlantic, New York, St Martin´s Press, XIV.

[22] Rummel J. R. and Heenan D. A. (1978). How Multinationals analyze Political Risk, *Havard Business Review*, January-February.

[23] Fry H. E. (1983). *The Politics of international Investment*, Mc Graw-Hill, New York.

[24] Graham M. E. and Krugman P. R. (1993). *The Surge in Foreign Direct Investment in the 1980s, in Foreign Direct Investment* ed. by Froot A. Kenneth, University of Chicago Press, Chicago.

[25] Strauch, R. (1980). Risk assessment as a subjective Process, The Rand Paper Series ed. by Rand Corporation, March 1980, pp. 4.

[26] Shapiro, A. (1978). Financial Structure and the Cost of Capital in the Multinational Corporation, *Journal of financial and quantitative Analysis* **13**: 211-266

[27] Shapiro, A.(1992). *Multinational financial Management* 4ed., St Martin's Press, New York.

[28] Samuelson, L. and Bond, E. W. (1986). Tax Holidays as Signals, *American Economics Review* **76**: 820-826

[29] Mahajan, A.(1990). Pricing Expropriation Risk, Financial Management, Winter 1990, p. 77-85

[30] Raff, H. (1991). *A Model of expropriation with asymetric Information*, Working Paper Nr 9105, Department of Economics, Université Laval.

[31] Clark, E. (1997). Valuing Political Risk, *Journal of International Money and Finance*, **16**(3): 477-490

A Step Ahead To Connect Your Bus-With GPS

A. Srikanth Reddy Nagadapally, B. BalaKumar Krishnaswamy, and C.Abhigna Reddy Shettap
D.Devireddy Prathika E.Roger Lee
Computer Science, Central Michigan University, Mount Pleasant, Michigan, USA

Abstract - Tracking of bus becomes an essential part in the process of bus application as the buses maybe delayed due to certain reasons like weather. Many techniques are used for tracking the bus but few of these have tracked only where the bus is located in the particular bus stop and none have attempted to track the exact location of the bus. In this paper, we present a technique to track the accurate location of the particular bus in which the passengers has booked his ticket. We report results of an experiment comparing this new technique to the existing technique for tracking the bus, assessing both the adequacy of the new technique and their ability to detect the faults of an old technique for tracking a bus. Our results show that tracking a bus through GPS can give an accurate location more effectively than those produced by messaging, making call techniques considered; however, the faults detected by the two techniques differ, suggesting that the techniques are complementary.

Keywords: GPS, Bus Tracking, Real Time Location, Bus Tracking through GPS, Smart bus tracking app, Global Positioning System.

1 Introduction

Public transport has been one of the most important modes of transport for people and much need to be done to improve some quality aspects of this service, especially in buses. While there are lot of inflight options in buses like Wi-Fi, air-conditioning etc, in-contrast a passenger standing in bus stop doesn't have many facilities apart from the bus shelters. For example, on a stormy day there may be more traffic than usual and he might not know about the delay in the bus arrival. For this purpose, it becomes essential to integrate additional features in this mode of transport. Tracking of bus becomes an essential part in the process of bus application as the buses maybe delayed due to certain reasons like weather. So far tracking is done on messaging level, personal calling and other basic levels. On a messaging level, it is done using a VMD (Variable Message Display) which tracks the coordinates of the bus and sends this information to the Server. By checking the passenger's needs, a notification indicating the arrival time of the bus is broadcasted to GSM module of all the passengers who have subscribed for similar type of service. The effectiveness of foregoing methods and techniques have been evaluated only in terms of ability to achieve the location of the bus with low accuracy; in our search of literature, we find no reports on studies assessing the effectiveness of techniques in
terms of ability to achieve the current or accurate location of the bus.

In this paper, we wish to merge GPS tracking with the mobile application so that we can get the exact location of the bus with more accuracy by showing the current location of the buson a map and reducing the dependability on other less accurate techniques and save the passenger his time and other resources. However, in extreme conditions where there's lack of data in mobile and a downloaded version of the maps is not available, the tracker may not be able to share the exact location of the bus and it works only on select platforms of Android and iOS.

2 Background and Related Work

Here the sequence diagram clearly shows the flow of work for booking a ticket in online. The passenger searches for a bus that goes to a particular destination then the booking system retrieves the information of that bus and displays the result to the passenger. Next the passenger checks the cost of the ticket which varies depending on the luxuary provided by the bus, this information is provided by the booking system and after that in order to book the ticket thepassenger has to login into his account by entering the credentials. Here the booking system forwards the credentials to passenger database where it will verify whether the credentials are matching with the database and allows the passenger to enter into his account. Then the passenger enters the details of the passenger where the details will be stored in the database. After making the payment booking system will show the summary of the booking details and the reservation has been stored in the reservation system and the passenger gets the confirmation that the ticked is booked. [5]

With the arrival of mobile platforms of Android and iOS, technology has advanced in such a manner allowing the user to book his tickets using his mobile application. The interaction between user and mobile application starts with the mobile data connection that the user has on his phone (WiFi or GPRS). The background process that occurs is that the request is sent to the webserver via the mobile gateway where the system based on the passenger request will decide the use of either the application server or the database systems. For

example, since ours is a project on bus app, all the interactions will be handled by the application server and all the

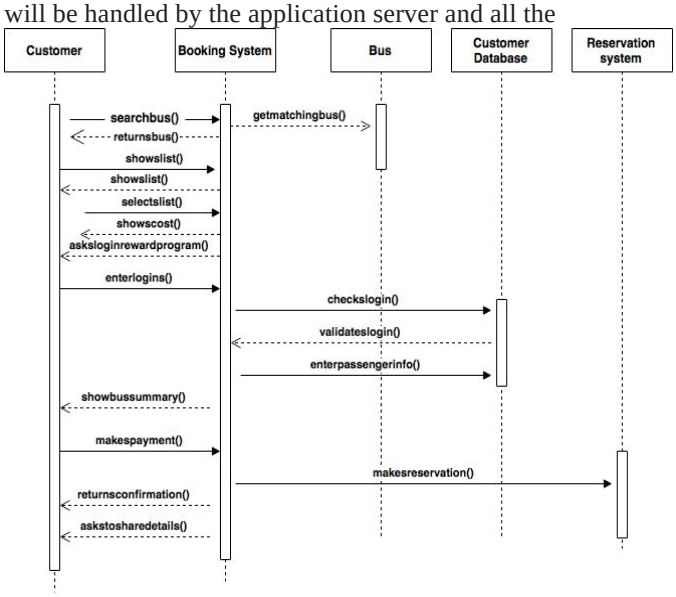


Fig .1.Sequence diagram for bus ticket reservation system [1]

information that is to be stored will be done by database, like storing information about passenger details. For an application having the bus information system, the different modules for instance might be Sign In, Booking of a trip, Station locator, Details of our Trips, Travel information and Rewards are the existing features used by the application. [3]

Other requests are more complicated and require additional processing which leads to more complex classes interacting with other applications. User provides data primarily through forms consisting of input fields such as text boxes, check boxes, selection list rendered in the mobile app. Information is translated in to a set of name-value parameters and becomes part of the request.

Client-Server

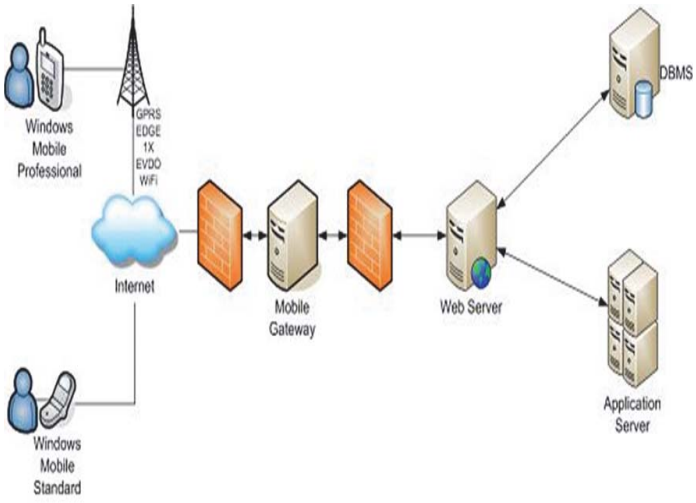


Fig.2. General Interaction between the user and the system [2]

Related work on bus tracking

Tracking of bus becomes an essential part in the process of bus application as the buses maybe delayed due to certain reasons like weather. So far tracking is done on messaging level, personal calling and other basic levels. On a messaging level, it is done using a VMD (Variable Message Display) which tracks the coordinates of the bus and sends this information to the Server. By checking the passenger's needs, a notification about the bus with the time is broadcasted to GSM module of all the passengers who have subscribed for similar type of service. [3]The effectiveness of foregoing methods and techniques has been evaluated only in terms of ability to achieve the location of the bus with low accuracy; in our search of literature, we find no reports on studies assessing the effectiveness of techniques in terms of ability to achieve the current or accurate location of the bus. [4]

In this paper, we wish to merge GPS tracking with the mobile application so that we can get the exact location with more accuracy by showing the map of the bus in its current location and reducing the dependability on other less accurate techniques and saves the passenger his time and other resources. However, in extreme conditions where there's lack of data in mobile and a downloaded version of the maps is not available, the tracker may not be able to share the exact location of the bus and it works only on select platforms of Android and iOS.

Spireon also offers four GPS tracking systems for companies managing fleets of vehicles. Fleet Locate Trailer and Asset Management is customized for companies looking to track their trailers using our rich, real-time data and analytics around trailer utilization and management; Fleet Locate Enterprise Fleet Management solution is designed for companies managing enterprise sized fleets of 500 vehicles or more in their fleet; Fleet Locate Local Fleet Management solution has been developed for the smaller to med-sized local and regional businesses needing a simple and affordable solution; and finally, Vehicle Path is Spireon's GPS tracking solution for small to medium size businesses looking for personal, localized support in their neighborhood and is sold exclusively through Spireon's authorized reseller channel. [8]

The GeoZigBee wristwatch device was designed specifically to provide anultra-low-power, miniaturized wireless GPS tracking device. It includes a low-power 2 GPS sensor, a flash memory and a ZigBee wireless data Link. This web-based architecture allows the common infrastructure to be leveraged by a variety of different users. In this paper, we describe an open architecture development approach that allows the Locator-Net server to deliver customized Location-based services using a combination of custom and publicly available data product. [5]

3 Methodlogy

3.1 Overview of Existing Technology

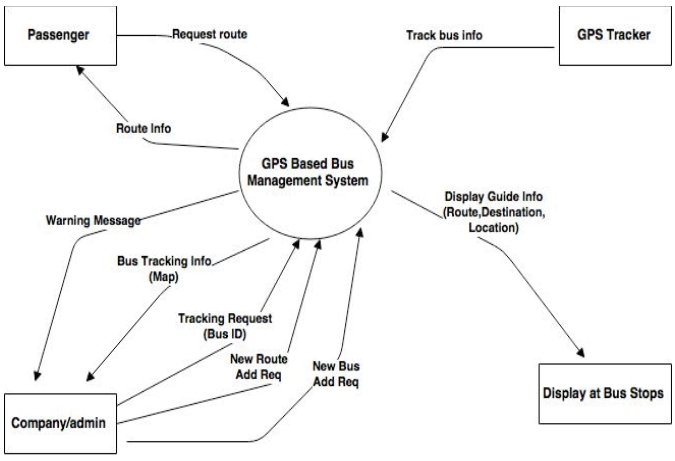
Public transport has been one of the most important modes of transport for people and much needs to be done to improve some quality aspects of this service, especially in buses. While there are lot of inflight options in buses like Wi-Fi, a passenger standing in bus stop doesn't have many facilities apart from the bus shelters. For example, on a stormy day there may be more traffic than usual and he might not know the delay. For this purpose, it becomes essential to integrate additional features in this mode of transport. [4]

3.1.1 Summary

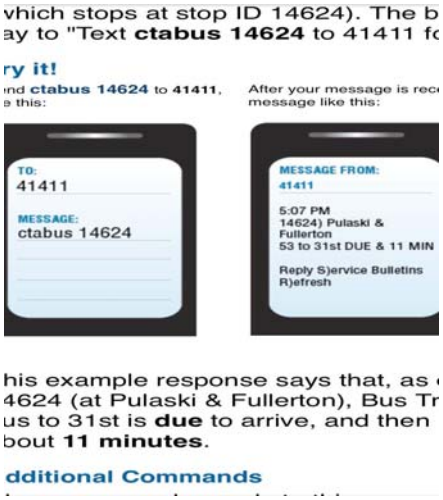
Tracking of bus becomes an essential part in the process of bus application as the buses maybe delayed due to certain reasons like weather. So far tracking is done on messaging level, personal calling and other basic levels. On a messaging level, it is done using a VMD (Variable Message Display) which tracks the coordinates of the bus and sends this information to the Server. By checking the passenger's needs, a notification about the bus with the time is broadcasted to GSM module of all the passengers who have subscribed for similar type of service.The message contains the location name where the bus is located, but there will be no correct information regarding when the bus may arrive to the destination. [4]

The effectiveness of foregoing methods and techniques has been evaluated only in terms of ability to achieve the location of the bus with low accuracy; in our search of literature, we find no reports on studies assessing the effectiveness of techniques in terms of ability to achieve the current or accurate location of the bus.

Dataflow Diagram



Data flow diagram of existing system [6]



Output of existing methodology [11]

3.2 Approach of New Technology

The foregoing techniques are strictly message based wherein the information provided is not up to date and not informative. On researching about these deficiencies, we decided to come up with a solution to integrate a GPS tracking system for buses. Our research deals with vendor specific application for the bus.

Overview of Intended Research

Generally, passengers nowadays have the option of booking tickets through their mobile application. Payment is done online and ticket is generated with a unique number specific to the ticket. Now if we integrate the application with an extra feature to track your bus, passenger can login to the application, select his bus and click on the GPS tracking option. This will launch the GPS tracking which will show the exact location of the bus in the mobile application. Global Positioning System helps us find the location of something that we are searching for. It is used by vehicles like cars, cell phones etc. The GPS device has a receiver which receives information from the special satellite signals. GPS can also track the speed/velocity and time of the moving object. The units keep an eye on and send\receive data from multiple satellite signals so as to give us better accuracy. So, if the carrier of the GPS has inbuilt or downloaded maps, it just needs the signals, else it uses the help of external factors like WiFi or GPRS data to download the maps and use it for signal. [2] Here the bus is set to be used as source to be tracked and the target will be information transfer of bus' location to the passenger's application. GPS tracking system greatly enhances the existing text based location tracking system and it also complements the application

3.2.1 Architecture of Improved System

Architecture of improved system

3.2.2 Implementation

Modules Used

Login: This modules appears in the first page. When the user initially opens the application he is prompted to enter his User-name or e-mail id and Password and then click the login button, so that he is successfully logged-in.

Register: Register appears on the first page and when clicked we get redirected to a page asking for few credentials, here both the passenger as well as the driver who are first time users can register.

(i)Passenger: The passenger who is a first time user will have to give details such as his First-name, last name, email-id,password, confirm password and select the type as student, which means he is registering as a new passenger.

(ii)Bus driver:Initially a new driver will have to register to use this application, he will have to give his details such as his first name, last name, email-id,password, confirm password and in the type drop down he should select driver, which means he is registering a new bus.

Search: This module is used to search for the buses ,where the list of buses depending on the bus numbers are displayed and user will select the bus number that he has booked the ticket for and then he can view a map which shows the current location of the bus as well as the passenger.

3.2.3 Testing

Functional Testing

In functional testing basically the testing of the functions of component or system is done. It refers to activities that verify a specific action or function of the code. Functional test tends to answer the questions like “can the user do this” or “does this particular feature work”. This is typically described in a requirements specification or in a functional specification.

The techniques used for functional testing are often specification-based. Testing functionality can be done from two perspectives:

Requirement-based testing: In this type of testing the requirements are prioritized depending on the risk criteria and accordingly the tests are prioritized. This will ensure that the most important and most critical tests are included in the testing effort.

Business-process-based testing: In this type of testing the scenarios involved in the day-to-day business use of the system are described. It uses the knowledge of the business processes. For example, a personal and payroll l system may have the business process along the lines of: someone joins the company, employee is paid on the regular basis and employee finally leaves the company.

Performance Testing

Performance testing, a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.

Performance Testing Techniques:

Load testing - It is the simplest form of testing conducted to understand the behaviour of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc., are also monitored.

Stress testing - It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.

Soak testing - Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.

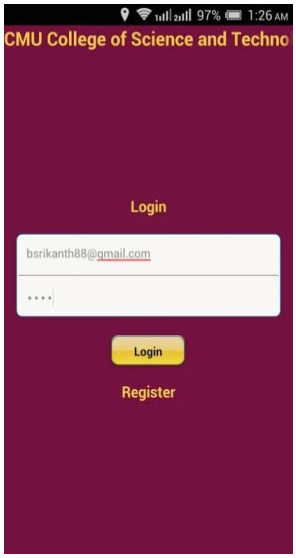
Spike testing - Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the workload.

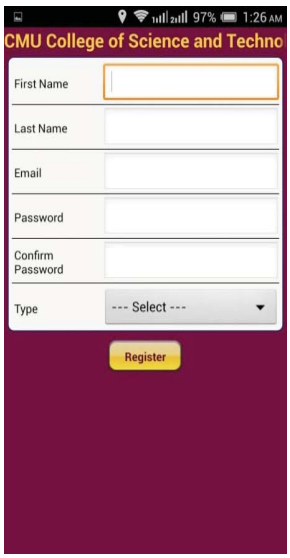
3.2.4 Results

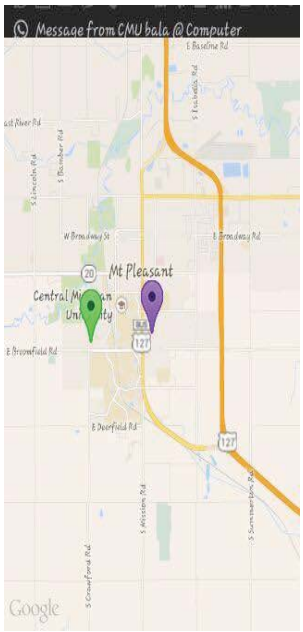
In the existing approach, the location is ambiguous. Whereas in our approach, the location will be specific and as it pinpoints the exact location. Now the user can now get the exact location of the bus if it is not on time or other delays. It is a value addition, which will greatly enhance the passenger experience for the people travelling in buses.

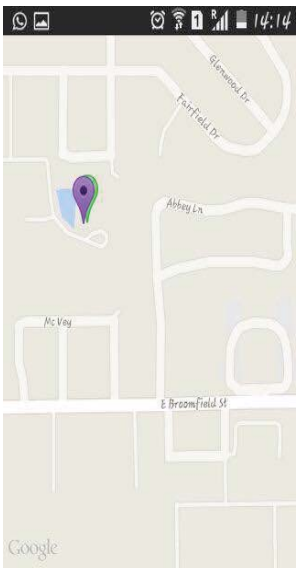
4 Analysis of Results

The results of the proposed system bring us a new approach to get away easily. By considering all the attributes into consideration and by comparing with all the methods the results in our approach are pretty much better than the previous system. Previously we can track the bus by only message protocol with low accuracy and in the present system tracking became very easy with more accurate results, the output of our results are shown below in the pictures









Representation of output in GPS tracking approach

5 Conclusion

We have presented a new approach for tracking the bus and several techniques for implementing that approach. This new approach differs from the existing system. We have presented the results of the controlled experiment suggest that this approach effectiveness is more formal than the existing system

The GPS tracking approach that we have presented have several additional potential advantages over other approaches. First, because this approach utilize passenger requests as a basis for generating a route map of the travelling bus which is major limitation of existing system. Second, the waiting time of passengers to receive a inaccurate location of the message is relatively small as they don't need to wait for message to receive. This is not the case with the new system where they can view the location of the bus in the map.

Finally, we believe that our results suggest that GPS Tracking could be used to address the problem with the existing system allowing passengers to access the accuracy of the proposed system.

6 References

[1] Urs, Chaitra N ; Chatterji, Sourindra ; SrivatsaSneha, M "A Mobile Application for bus notification system" pp. 724 – 727, 29 Nov. 2014

[2] Fleischer, P.B.; Nelson, A.Y.; Sowah, R.A.; Bremang, A., "Design and development of GPS/GSM based vehicle tracking and alert system for commercial inter-city buses," *Adaptive Science & Technology (ICAST), 2012 IEEE*

4th International Conference on , vol., no., pp.1,6, 25-27 Oct. 2012

[3] Chadil, N.; Russameesawang, A.; Keeratiwintakorn, P., "Real-time tracking management system using GPS, GPRS and Google earth," *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on* , vol.1, no., pp.393,396, 14-17 May 2008

[4] Michael, K.; McNamee, A.; Michael, M., "The Emerging Ethics of Humancentric GPS Tracking and Monitoring," *Mobile Business, 2006. ICMB '06. International Conference on* , vol., no., pp.34,34, 26-27 June 2006

[5] Mark J. Timm, Walter A. Dorfstatter "Vehicular emergency message system" pp.419-424 Nov 1996

[6] http://creately.com/blog/examples/sequence-diagram-templates/

[7] http://www.satter.org/2008/01/microsoft-mobil.html

[8] http://www.spireon.com/how-gps-tracking-works

[9] http://vast.uccs.edu/~tboult/PAPERS/ion-07-gps-tracking-wristwatch.pdf

[10] http://www.slideshare.net/neerajkansal7/neerajatulankit

[11]http://www.transitchicago.com/riding_cta/how_to_guides/bustrackertext.aspx

PRONTO System: integration between doctors and pharmacists in the basic health care

Mauro Marcelo Mattos, Luciana Pereira de Araújo, Maria Eduarda Demmer, Shaiane Mafra Casa, Eric Boeing, Jonathan Rodrigues Ev, João Magnani, Simone Erbs da Costa, Marcio Michelluzzi, Caíque Reinhold, Rogério Mello Vanti, Jacques Robert Heckmann
Development and Technology Transference Lab, Computer System Department, University of Blumenau
Blumenau, SC, Brasil
mattos@furb.br, lpa@furb.br, medemmer@gmail.com, shaimafra@gmail.com, ericbenck@gmail.com, jonathanrodriguesev@gmail.com, jogamabnu@gmail.com, si.gen@terra.com.br, marciomichelluzzi@gmail.com, caiquereinhold@gmail.com, rmvanti@gmail.com, jrh@furb.br

Abstract - *This paper present a PRONTO system that is a management public health system deployment at Blumenau city. This system integrate the functionalities need to provide the assistant to the patients. In this paper we related the experience during the deployment with relation to integration between doctors and pharmacists. This integration is need because the doctors prescribe drugs and the pharmacists dispense them. Until this moment, it was made 25,191 prescriptions using PRONTO system in a total 28 units that use this system..*

Keywords: Health care; prescription; PRONTO system.

1 Introduction

The PRONTO System is a health care system developed in Blumenau city with goal to integrate the primary health care and secondary health care of the city [1]. Blumenau provides the public health care existing at Brazil. This system called "Sistemas Único de Saúde" (SUS), in English Health Unique System offers free health services to citizens, like medical consults, pharmacy, dental appointment, dispensing drugs, health proceedings and other [1,2]. This system was created with Brazilian Constitution at 1988 and has the goal to allow the free health care for all citizen [3,1].

Even though the name, in begin, the SUS was not computerized and the health care was not centralized [1,4]. Each Brazilian city has your system that can be computerized or not, and the same patient can have many health records [5]. This is possible because when the patient goes to other unity health care he/she does not bring his health record is created another health record for him.

This lack of centralization causes loss for health proceedings and for the health patient himself [4]. An example is when the patient go to doctor and the doctor make a prescription. In the next medical consultation, the doctor does not know if the patient took off the drugs at the pharmacy because the pharmacy do not has this information. And because the doctor did not communicate with the pharmacists. This is dangerous because the doctor can continue the treatment thinking that the patient took off the drug, but in really he did not took.

With the PRONTO System, the services provided by SUS are centralized and unified. The medical consultation, the pharmacy and other services are integrated and each one can see what the patient do and where the patient go in the PRONTO.

This paper presents the relationship between doctors and pharmacists through the PRONTO System from an experience report that occurs in Blumenau city. This paper follows. The related works section presents and discusses about other paper related to this. The PRONTO System section presents the system and its functionalities. The experience report section describes the used system by the health professionals and real patients in the unity health centers with focus between doctors and pharmacists. Finally, the discussion section present the results obtained with the experience and discusses about the results in a widespread case.

2 Related Works

In this section we present some paper related with our research.

Rigby et al. [6] relate about collaboration between doctors and pharmacists. They say that the relation between them during a prescription is important because the pharmacist works with the drugs and they know the benefits of each one, and in times, they know about the drugs more than doctors. Then, for the authors, the pharmacists should participate of medical consultations and help to make the prescriptions. This collaboration could be made by a system and the result for the patient will be most efficient.

Teixeira [4] describes the decentralization of SUS services and points the losses caused by this decentralization. For author the professional's health should communicate to study the patient case and solve a collaborative way his problems. Teixeira comment that the computerization each health unit made your system and the information is not centralized. For author, the decentralization services delayed healing problem of patient.

Lehmann et al. [7] discusses about the patient information shared among the health professionals. Although the patients knew that their information it were shared and they knew that is provide relevant information to treatment, some patient did not agree with this shared information. Other side, related to shared information with the pharmacists, the most patient agree with this shared. It is possible because the patient has more contact with the pharmacists and they know the pharmacists role because they dispense the drugs for them.

Baysari et al. [8] realized a study about the need alerts during drugs dispensation at pharmacies about intolerance and allergy that patient have. The authors perceive that the dispenser receive an excess of alerts that need analyzed to know if it is important or no. Then, the dispensers started to not read the alerts. In the study, the authors conclude that the use of alerts on a system must be done carefully so they are not fired irrelevant way leading to not reading the relevant information.

The papers present are related to this because they discuss the relation among health professionals, with focus in pharmacists and doctors. The differential of this paper is that we applied the research in real environment and we obtained a positive point with the information integration. We do not promote the direct collaboration among the professionals but in indirectly way they collaborate one with other.

3 PRONTO System

The PRONTO system was developed through agreement between Blumenau City Hall and University of Blumenau (FURB) by Development and Technology Transference Lab (LDTT). The PRONTO is a public health management system that allow to take decisions supported by updated information while that enables the service optimization processes to citizen [5,9].

The PRONTO integrate and computerize the public health, initially at Blumenau city [5,10]. Although the PRONTO architecture allow it deployment in other cities.

The system is developed from weekly meetings that involve system users, managers, system analysts and designers with intent to available the functionalities need to public health network at SUS in Blumenau [10].

The system also aims to reduce the quantity of care in units and hospitals with secondary and high complexity [1]. The system is divided in modules to facility the access by professionals that use it. These modules are: administrative, treatment, pharmacy, stock, management, health community agent and center for testing and counseling [1]. Each module is visible according profile user and it is possible configure what buttons will be enabled or no inside each one.

The doctor, for example, has allowed accessing the treatment module to make his medical consultations. In his screen, the doctor sees a list with patients that are in treatment line. Then, the doctor call a patient, make the evolution in his electronic health record, make the prescription, see the drugs that the patient took off in the pharmacy and all functionalities necessaries. Otherwise, the pharmacist has allowed to access the pharmacy module. In this module he can dispense drugs for the patient.

The PRONTO prescriptions has a barcode, then if the patient goes in a SUS pharmacy with a PRONTO prescription, the pharmacists write the barcode in PRONTO system and the dispense screen is filled. With this, the pharmacist can see better what the drugs the patient need than the manual prescription. Also, all dispersions were registered in the system, i.e., a patient cannot get the same drugs before duration these drugs write in the prescription.


IDENTIFICAÇÃO DO EMITENTE

RECEITA

43471557201400000021

PREFEITURA BLUMENAU

Unidade: UNIDADE TREINAMENTO 1
Telefone: 47 33210000
Usuário: ADMINISTRADOR DE SISTEMA

MEDICAMENTOS DO SUS

1. PARACETAMOL 500MG COMPRIMIDO.....30 unidades
Take one after wake up

2. ACIDO ACETILSALICILICO 100 MG COMPRIMIDO.....60 unidades
Take two pils for day after lunch

Figure 1 – Part of normal prescription by PRONTO

A sample of PRONTO prescription can be viewed in Figure 1. All prescription has a list with drugs. For each drug have a amount of pill and what time this amount will be last. Normally, the time corresponds to 30 days. The PRONTO generate four prescription kinds: normal, type A, type B, controlled and antibiotic. These are the kinds that SUS offers and are differentiated by time validation and drug types.

The PRONTO implements a unique health record, i.e., each patient has just one electronic health record in network. Thus, all professionals that use PRONTO can see what the treatment patient and all things that the patient through PRONTO. For reply the data patient to all health units we implement a replication networks. Each unit has one server

that replies the data patient for other units. Thus, each unit has the same data. In Figure 2 it is possible see this structure related to the electronic health record.

Figure 2 – Electronic health record architecture

4 Experience Report

The PRONTO development started in August 2011 and in June 2012 was installed in a health unit to be tested. In this health unit, the PRONTO already has been used to medical consultations.

The PRONTO was installed by deployment team of university. This team follows the health professionals during two weeks to help with the system use. Initially, the professionals had difficulties that were clear with the use. Related to the patients, some like the system and other do not. The good side of the system is the integration among the health services and the data integration. Other hand, some patients see the control of actions patient like some bad. This control implies that a patient just can take off drug at pharmacy if he did not get yet. And if the patient lack of a medical consultation, the system add a lack for this patient. And with this information the health professional can have a control of patient activities.

In the first health unit the system was tested and approved. Then, one year later, the PRONTO was installed in public pharmacies, in total nine pharmacies. Initially we have much confusion in the pharmacy because each patient has to register on the system to then take off his drugs. And, if the patient has already got the drug, he did not take off again.

After we test in one health unit and pharmacies we began deploy the PRONTO in other health unity. For each new unit had training in a laboratory at university with the health professionals to learn about PRONTO System. Next, the deployment team follows the professionals at unit during the treatments.

After three deployment years we have 28 health units integrated using the PRONTO. This represents two regions of the city and about 38% of all health units at Blumenau. These units are basic health care unit, general ambulatory, polyclinic, advice testing center (in Portuguese CTA) and university hospital (secondary treatment). All these units are integrated, i.e., if the patient goes in one unit, the other units can see what this patient did and where he went (this visualization is controlled by user profile).

With this integration, some doctors began to see the drugs that the patient uses through the system. This is amazing because some patient does not remember the drug name that uses and was other doctor that prescribes it. Then, the doctor can prescribe new drugs or just renew the last prescription by the system.

Initially, the doctor could view the drugs withdraw at the pharmacy but do not see the last prescriptions. This occurs because the system was deployed in parts, and some units do not have the system to make the prescription. But, just with the visualization of withdraw drugs; the doctors had a better information about the patient. The screen that represent the withdraw drugs can be viewed in Figure 3.

Data e hora	Cód. barras / Nº receita	Lote	Data da receita	Quantidade	Duração	Prazo	Dias restantes	Unidade	Dispensador	Prescritor	Conselho	Inscrição
22/12/2014 22:54	43471557201400000021	ACIDO ACETILSALICILICO 100 MG COMPRIMIDO	22/12/2014	60	30	21/01/2015	30	UNIDADE TREINAMENTO 1	ADEMAR WILD WACHHOLZ	ADRIANA TIENGO	CRM	5205
22/12/2014 22:54	43471557201400000021	PARACETAMOL 500MG COMPRIMIDO	22/12/2014	30	30	21/01/2015	30	UNIDADE TREINAMENTO 1	ADEMAR WILD WACHHOLZ	ADRIANA TIENGO	CRM	5205
24/10/2014 08:31		COMPLEXO B (POLIVITAMINICO) COMPRIMIDO REVESTIDO	24/10/2014	10	10	03/11/2014	0	BASE DADOS DEMONSTRAÇÃO - 1490	ADMINISTRADOR DE SISTEMA		CRM	

Figure 3 – Drugs get in the pharmacy

This screen is divided in columns that provide information about the drugs withdraw by the patient in a SUS pharmacy. During the visits in the health units, we observe that the most important information is how many days to complete treatment. This information is important because the doctor can know if the patient gets the drug in the pharmacy and the data. Therefore, he can know if the drug is taking effect as expected.

With the use system, the doctors feel the need to prescript drugs that are external to SUS. In other words, the SUS offers for free some drugs that are more used by citizen. And these drugs normally are generics. The doctors can prescribe drugs who did not exist in SUS and who need buy in a pharmacy. Initially, for this kind of drug, the doctor has been prescribing in a manual form. For solve this impropriety we developed a prescription without SUS drugs. Thus, if the doctor wants prescribe drugs external SUS he can. This prescription is made on white field and is printed like the other prescriptions.

4.1 About Numbers

Until November 2014, the PRONTO System generated 25,191 prescriptions, and of these prescriptions 12,262 were for different patients.

Related to kind prescription the Table 1 relates the kind prescription and the amount prescript of this kind. This shows

that the most of prescriptions are normal. This is acceptable because the normal prescription offers the common drugs like aspirin and *paracetamol*. Drugs to treat headache, stomach ache and other common diseases. Type A and type B prescription are less frequent because this prescription offers the drugs more dangerous and are most controlled than normal. And this kind prescription need a number provided by national health surveillance agency (ANVISA), i.e., the doctor must have the prescription in a specific form and, if he want, can prescript by PRONTO too for registered in the system.

Table 1. Prescriptions by kind

Kind	Amount
Normal	16,853
Type A	3
Type B	602
Controlled	3,102
Antibiotic	4,631
Total	25,191

Analyzing the PRONTO prescriptions we observed that 4,137 prescriptions do not have drugs SUS, i.e., this prescriptions offers just drugs who need buy at a common pharmacy. This information was taken looking at the prescriptions that had only the field of external drugs SUS filled.

5 Discussions

With the computerizing of the health network Blumenau was possible integrate the services provided by doctors with the pharmacists' services. The most important service integrated was the prescriptions. With this integration the pharmacists can read and controlled better the dispensing drugs because the prescriptions are digitalized. The pharmacists can see how much drugs the patient took off at pharmacy and what drugs he use. Also, the pharmacists have a access to history of dispensations made to patient and can control if the patient already have the drug.

In other side, with the computerizing, the doctors have the integration of electronic health record patient, prescriptions and can see what drugs the patients withdrew from the SUS pharmacy. This information helps the doctors to control the treatment used to care the patient.

Although the experience, it is possible conclude that with computerizing and the integration of health services, the professionals health can control better the treatment patient because they have information of all units that the patient went. They can see the problems that occur with the patient and see the drugs that he takes, even without the need to give this information.

With future works, we will continue deploy the PRONTO System at Blumenau city, and next, we will pretend deploy in other regions to integrate many cities with the same system. Thus, the same electronic health record could be viewed by professional health of many cities.

6 References

[1] Araujo et al., 2014. PRONTO: An integrated health care system for Blumenau city. WWW/Internet, IADIS, 2014.

[2] Araujo, L. P.; Berkenbrock, C. D., Mattos, M. M., 2014. Using participatory design in designs phase of collaborative system. CSCWD, 2014.

[3] Saúde, M. da, 2006. Entendendo o SUS. In Portal da Saúde, Governo Federal, pp 5-7.

[4] Teixeira, R. R., 2009. Humanização: transformar as práticas de saúde, radicalizando os princípios do SUS. In Interface-Comunicação, Saúde, Educação, Vol. 13, No. 1, pp 785-789.

[5] Mattos, M. M. et al, 2013. Sistema de Informação Ubíquo na Gestão de Saúde Pública. In Saúde: a contribuição da extensão universitária, Univille, Vol. 1, 1st Ed.

[6] D. Rigby, "Collaboration between doctors and pharmacists in the community," Australian Prescriber, vol. 33, no. 6, pp. 191-193, 2010.

[7] Lehnbom et al., 2013. A Qualitative Study of Swedes' Opinions about Shared Electronic Health Records. MEDINFO, 2013.

[8] Baysari et al., 2013. Identification of strategies to reduce computerized alerts in an electronic prescribing system using a Delphi approach. MEDINFO, 2013.

[9] P. M. de Blumenau. (2010) Terceira idade - fundao pro-familia. Prefeitura Municipal de Blumenau. [Online]. Available: http://www.blumenau.sc.gov.br/gxpsites/hgxpp001.aspx? 1,19,294,O,P,0,MNU;E;117;2;MNU;,

[10] LDTT. (2012) Pronto gestao de saude publica. Laboratorio de Desenvolvimento e Transferencia de Tecnologia (LDTT) - Universidade Regional de Blumenau (FURB). [Online]. Available: http://www.furb.br/ldtt/projetos/pronto-gestao-de-saude-publica.

SESSION

MODELING LANGUAGES, UML + PETRI NETS, SOA AND APPLICATIONS + PORTABILITY + REQUIREMENTS ENGINEERING and ASPECT ORIENTED SOFTWARE ENGINEERING

Chair(s)

TBA

Modeling Elevator System With Coloured Petri Nets

Mohammed Assiri, Mohammed Alqarni and Ryszard Janicki
Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada L8S 4L8

Abstract—*A fairly general model of the elevator system is presented. Coloured Petri Nets (CPN) and CPN tools are adopted as modeling tools. The model, which is independent of the number of floors and elevators, covers different stages of the elevator system in substantial detail. The model assists simulation-based analysis of different algorithms and rules which govern real elevator systems. The results prove the compatibility and applicability of this model in various situations and demonstrate the expressive power and convenience of CPN.*

Keywords: Formal Specification, Elevator System, Software Specification Benchmarks, Coloured Petri Nets

1. Introduction

The elevator system is one of the software engineering benchmarks which are frequently used to test the expressive power, readability, and convenience of various formal specification techniques [1]. Petri Nets is one formal specification technique.

In [2] and [3], dynamic scheduling of the elevator system was modeled by Petri Nets, and hybrid Petri Nets. Timed Petri Nets, Abstract Petri Nets and Elevator Control Petri Nets were used in [4], [5], and [6] respectively. Furthermore, the elevator system was modeled by Coloured Petri Nets in [7], and Timed Coloured Petri Nets in [8] and [9].

Nevertheless, all of these previous models are either static or dependent on a particular number of elevators and floors (often one place was required for each elevator car), the concept of colour as a data type was not fully utilized, or other formalisms such as UML were substantially involved.

Our model is independent of the number of floors and elevators and covers different stages of the elevator system in substantial detail. We believe our model is flexible enough to be adapted to different algorithms and rules, and may eventually evolve into a 'standard' formal model of the elevator system.

2. The Elevator System

Elevator systems are an integral aspect of buildings from the point at which they are first designed. With high-rise buildings being the typical candidate for elevator systems, such systems are usually very complex. Multiple elevators must be controlled by a centralized control mechanism. The complexity of these elevator systems arises from factors such as scheduling needs, resource allocation, and stochastic control, to name a few. Handling these jobs usually results in systems behaving as discrete event systems [10].

The elevator system is usually defined as follows [1]: An *elevator system* is to be installed in a building with m floors and n cars. The elevator and the control mechanisms are supplied by the manufacturer. The internal mechanism of an elevator system is assumed (given). The problem concerns the logistics of moving cars between floors according to the following constrains:

a. Each elevator's car has a set of buttons - one for each floor. Pressing these buttons signals the elevator to move to the corresponding floor.

b. On the wall outside the elevator each floor has two buttons (with the exception of the ground and the top floors). One button is pressed to request an upward moving elevator and another button is pressed to request a downward moving elevator. If both buttons are pressed, then each direction is assigned to a different car.

c. When an elevator has not received any requests for service, it should be held at its parking floor with its doors closed until it receives further requests.

d. All requests for elevators from floors (i.e. hall calls) must be serviced eventually. The applied algorithm controls the priority of floors.

e. All requests for floors within elevators (i.e. car calls) must be serviced eventually, with floors usually serviced sequentially in the direction of travel.

f. Each elevator's car has an emergency button which when pressed causes an alarm. The elevator is then deemed "out of service". Each elevator has a mechanism to cancel its "out of service" status.

Our model is based on the above description.

3. Coloured Petri Nets

Coloured Petri Nets (CPN), first proposed in [11] and later substantially modified and enhanced in [12], are an extension of Petri Nets which are often used to model behaviours of rather complex systems. CPN have preserved the useful properties of Petri Nets while at the same time extending the initial formalism to allow for distinction between tokens. Coloured Petri Nets (CP-nets or CPNs) is a graphical language for constructing models of concurrent systems and

analysing their properties. CP-nets is a discrete-event modelling language combining the capabilities of Petri Nets with the capabilities of a high-level programming language. Petri Nets provide the foundation of the graphical notation and the basic primitives for modelling concurrency, communication, and synchronisation. Coloured Petri Nets allow tokens to have a data value attached to them. This attached data value is called token colour. Although the colour can be of any arbitrarily complex type, places in CPNs usually contain tokens of one type. This type is referred to as the colour set of the place.

A semi-formal definition can be given as follows:

A *Coloured Petri Net* is a tuple:

$$N = (P, T, A, \Sigma, C, N, E, G, I)$$

where:

- P is a set of places.
- T is a set of transitions.
- A is a set of arcs.
- In CPNs sets of places, transitions, and arcs are pairwise disjoint $P \cap T = P \cap A = T \cap A = \emptyset$
- Σ is a set of colour sets defined within CPN model. This set contains all possible colour, operations, and functions used within CPN.
- C is a colour function which maps places in P into colour in Σ.
- N is a node function which maps A into $(P \times T) \cup (T \times P)$.
- E is an arc expression function which maps each arc $a \in A$ into the expression e. The input and output types of arc expressions correspond to the type of nodes which the arc is connected to.
- G is a guard function which maps each transition $t \in T$ into guard expression g. The output of the guard expression should evaluate to Boolean value true or false.
- I is an initialization function which maps each place p into an initialization expression i. The initialization expression must evaluate to a multiset of tokens with a colour corresponding to the colour of the place $C(p)$.

CPN support hierarchical modeling and are equipped with a modeling language called CPN ML which is based on the standard functional programming language ML. There are a variety of tools that can be used. In this paper the tools from [13] have been used.

For more details and theory of CPN, the reader is referred to [14].

4. CPN-based Modelling of Elevator System

Due to the complexity of the elevator system and the desired flexibility of the structure, the proposed model is composed of five major interconnected but independent sub-models. These sub-models include the *car-structure sub-model*, the *hall-call sub-model*, the *car-call sub-model*, the *system-cycle sub-model*, and the *hierarchical parking-optimizer sub-models*. The functions and connections between sub-models are described as follows: The car-structure sub-model represents the elevator's cars. It is at the centre of all other sub-models that concurrently control the elevator's cars. Typically, an elevator car is requested by two types of controls: either a hall-call or a car-call. As the names suggest, a hall-call is placed by pressing a button located in the hallway of a given floor while a car-call is place by pressing a button inside the car of the elevator. When a hall-call is placed, by relying on algorithms the hall-call sub-model will assign the hall-call to the appropriate car of the car-structure sub-model. Similarly, the car-call sub-model coordinates the placed car-calls with the cars of the car-structure sub-model. The system-cycle sub-model operates the cars of the car-structure sub-model to service the requested calls. Finally, the parking-optimizer sub-models reduce the waiting time between the placing hall-call and the arrival of the assigned car by constantly electing the holding floors of the idle cars.

4.1 Car-Structure Sub-Model

This sub-model (Figure 1) consists of just two places *Cars* and *Database* that also belong to other sub-models. The first place has the colour set **Cars**, which is a record colour set or the Cartesian product of the sets described in Table 1. The second place has the set of colours **Database** (defined in Table 2). In principle, this is a list of all the necessary information about the states of cars. This list is used by the algorithms of the hall-call sub-model. Both places are initialized dynamically by the functions *initialize cars* and *initialize database* respectively.

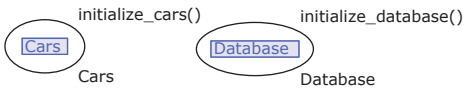


Fig. 1: The Car-Structure Sub-Model

Table 1: The definitions of colour set **Cars**

Colour Sets	Definitions
Car ID	$\{i \mid i \in \mathbb{Z}^+ \wedge i \leq total\ number\ of\ cars\}$
Range	$\{r \mid r \in \mathbb{Z}^+ \wedge lowest\ floor \leq r \leq highest\ floor\}$
Status	{*up, down, emergency, idle, out of service*}
Desired Floors	$\{[l] \mid l \in Range\}$
Call Issuer	{*request, system, non, reservation*}
INT	$\{n \mid n \in \mathbb{Z}\}$
Cars	{ *(car id, current floor, status, parking floor, desired floors, call issuer) | car id ∈ Car ID, current floor ∈ Range, status ∈ Status, parking floor ∈ Range, desired floors ∈ Desired Floors, call issuer ∈ Call Issuer*}

Table 2: The definition of colour set **Database**

Colour Sets	Definitions
Car Info	{*(current floor, status, destinations, car id)* | *current floor* ∈ *Range, status* ∈ *Status, destinations* ∈ *Desired Floors, car id* ∈ *Car ID*}
Database	$\{[d] \mid d \in Car\ Info\}$

In the colour set **Cars**, the *parking floor* indicates which floor the car is held when idle. The initial value of the parking floors is calculated in general by the following equations:

$$\begin{gathered} \textit{Floor No.} = \textit{(highest floor no.} - \textit{lowest floor no.)} + 1 \\ \textit{Scope} = \mid \textit{floors no.} \div \textit{numbers of cars} \mid \\ \textit{Scope's Head} = \textit{((scope * car id)} - \textit{(scope} - \textit{1)} + \\ \textit{(lowest floor no.} - 1\textit{))} \\ \textit{Scope's Tail} = \textit{(scope} * \textit{car id)} + \textit{(lowest floor no.} - 1\textit{)} \end{gathered}$$

Thus, a parking floor of a car is assigned optionally by either Scope's Head or Scope's Tail. Otherwise the parking floor may be typed manually for each car, especially in cases when the above equations are impractical.

The other elements of the colour set **Cars** are self-explanatory.

4.2 The Hall-Call Sub-Model

This sub-model assigns a hall-call to the most appropriate car based on the applied given algorithms (which are subject to changes and replacements). Furthermore, the model generates hall-calls from arbitrary floors and a selected floor in order to facilitate efficiently the examination of various rules and algorithms during the simulation-based analysis.

The processing of hall-calls is initialized from place *requested call* where each token represents a placed hall-call of colour set **Hall Call**. Every token has an appropriate direction and a floor number where the hall-call was placed. Assigning a hall-call to a car requires the firing of transition *Assign Hall Call*. Transition *Assign Hall Call* is enabled if and only if its guards, which represent appropriate rules, are holding. The specific rules that must be satisfied are comprised of the following:

1) the selected car is either idle or traveling toward the direction of the hall-call;
2) the selected car is not reserved; and
3) the selected car is elected by the applied algorithm.

After firing transition *Assign Hall Call*, the token of a placed hall-call is removed from place *requested call* and assigned to *the desired-floors list* of a selected car in place *Cars* with a guided direction, i.e. up or down, if the selected car is idle.

Two algorithms - namely the *nearest-car algorithm* [15] and the *scope algorithm* which process the assignment of hall calls to cars - are implemented separately to examine the model's ability of adopting various algorithms and rules.

Place *Database* facilitates the adoption of multiple different algorithms that require simultaneous access to all cars' states; hence, other algorithms can easily be adopted.

Table 3: The definition of colour set **Hall Call**

Colour Set	Definition
Hall Call	{*(hall call floor, status)* | *hall call floor* ∈ *Range, status* ∈ *Status*}

The nearest-car algorithm starts by analysing the token of place *Database* from Table 2. First, the cars with proper status (i.e. cars that are travelling toward the hall call request or that are servicing no calls) are extracted from the token. Each car is represented by a single tuple, so selection of cars occurs by extracting appropriate tuples. Once the proper cars are elected, the distances between the hall-call floor and cars' current floors are calculated by the absolute value of the difference between current floors and the hall-call floor for each car. Accordingly, the hall-call is assigned to the car with the minimum distance to the hall-call floor. Additionally, in this paper we have improved the nearest-car algorithm by further calculation of time consumed by the car's stops between the hall-call floor and the car's current floor. Thus, travel times plus the number of served calls between car's current floor and the hall-call floor are calculated for each car. Based on this, the car with the expected minimum waiting-time is assigned to serve the hall-call.

The scope algorithm is usually employed in express elevators and sky-lobby floors where each car is forced to serve a specified range of floors with an allowance of transit floors. We implement the scope algorithm as extra guards on transition *Assign Hall Call*. For instance, a guard that identifies the range of floors for each car is written as:

$$H \leq A \leq T.$$

where:
H = *the head floor of the car's scope*
A = *the answered hall-call floors*
T = *the tail floor of the car's scope*

The hall-call model also allows for a simulation-based analysis of different algorithms by controllably producing two classes of floors' numbers: arbitrary, where numbers range from lowest to highest floors; and an identified number of a specific floor that is requested repeatedly. Some parameters (in Table 4) are defined for controlling the production of hall calls. Moreover, a produced floor's number is associated with a direction based on the two rules. First, a floor's number equates the highest floor that is associated restrictedly with the down direction. Conversely, a floor's number equates the lowest floor that is associated restrictedly with up direction. The other floors' numbers are associated non-deterministically (modeled as a uniformly distributed random choice) to upward or downward direction.

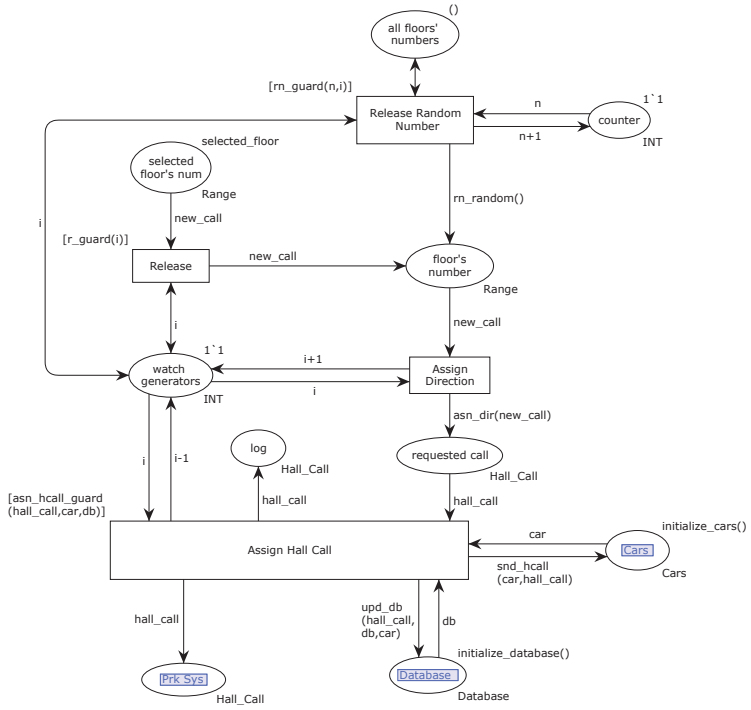


Fig. 2: The Hall-Call Sub-Model

Table 4: The Parameters of The Hall-Call Sub-Model

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite hall calls	$\{y \mid y \in \mathbb{Z} \wedge 0 \leq y\}$
The most requested floor	$\{r \mid r \in Range\}$
Duplication of requested floor	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$
The applied algorithm	$\{minimum\ waiting, nearest, scope\}$
Production's pause number	$\{p \mid p \in \mathbb{Z}^+\}$

4.3 The Car-Call Sub-Model

This sub-model provides a coordination between the cars and the car-calls. Additionally and similarly to the hall-call sub-model, this sub-model includes generators for both arbitrary and identified floors' numbers.

The coordination between cars and requested car-calls are represented by tokens in place *car call* with the colour set **Range**, which is a floor number in the range between the lowest and the highest floor. Placing the car-call in a car demands firing transition *Coordinate* which is enabled when its guards are satisfied in respect to the producing mode's state and the applied algorithm on the hall-call sub-model. For instance, in the scope algorithm the car serves only within the floors of the car's defined scope. After firing transition *Coordinate*, the placed car-call is removed from place *car call* and inserted into the car's desired-floors list with an appropriate direction if the car is idle. Similarly, the list of the specified calls, in place *specific floors' num* by colour set **Specific Floors** (see Table 5), is also merged.

Table 5: The definition of colour set **Specific Floors**

Colour Set	Definition
Specific Floors	*{(car id, specific calls, repeated times) | car id ∈ Car ID, Specific calls ∈ Desired Floors, repeated times ∈ ℤ}*

The car-call model also features two mechanisms that produce arbitrary car-calls where each call is placed individually into a car and a list of specified calls are placed entirely to each available car by transition *Coordinate*. Table 6 outlines some of the parameters of car control.

Table 6: The Parameters of The Car-Call Sub-Model

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite car calls	$\{x \mid x \in \mathbb{Z} \wedge 0 \leq x\}$
Most desired floors	$\{[f] \mid f \in Range\}$
Frequency of desired floors	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$
Production's pause number	$\{p \mid p \in \mathbb{Z} \wedge 0 < p\}$

4.4 The System-Cycle Sub-Model

This sub-model deals with the system cycle of the elevator's cars during the operation of the elevator system. Each elevator's car experiences three separate stages of maintenance, arrival, and transition (see Figure 4). Furthermore, the system-cycle sub-model has basic parameters which are very

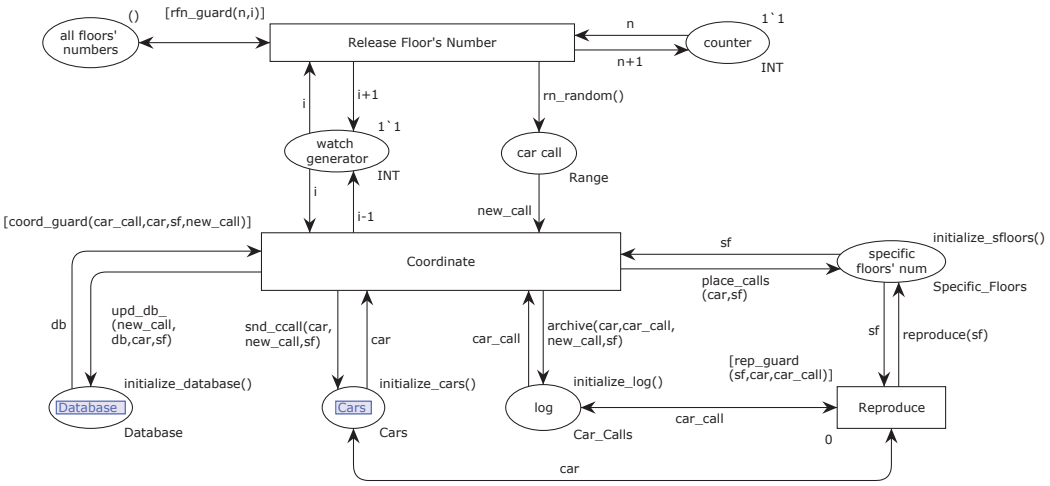


Fig. 3: The Car-Call Sub-Model

convenient for simulation-based analysis (defined in Table 7).

Table 7: The Parameters of The System-Cycle Sub-Model

Parameter	Legal value
Car's number	$\{i \mid i \in \mathbb{Z}^+\}$
Lowest floor number	$\{n \mid n \in \mathbb{Z}^+ \wedge n < highest\ floor\}$
Highest floor's number	$\{m \mid m \in \mathbb{Z} \wedge lowest\ floor < m\}$
Restart cars automatically	$\{yes, no\}$

The maintenance stage models the suspension of a car which is caused either by an emergency case when the car's emergency button is pressed, or by an operation failure case when an error occurs during the execution of the elevator-system model. The suspension is the result of firing transition *Maintain* which has the highest priority in the entire model (i.e. when it is enabled, all other transitions in the model are blocked). Therefore, when the status of a car is "emergency" or "out of service", then transition *Maintain* fires and the car's token is transferred temporarily from place *Cars* to place *out of service*. At this point the token in place *Database* is updated accordingly. Hence, the car is not accessible by any other sub-models that have no access to place *out of service*. Later, a pending car can be restarted either automatically or manually based on the value of parameter *restart cars automatically*. If it is assigned to "yes", then transition *Restart* is enabled immediately. However, if the parameter is set to "no", then restarting pending cars requires manually altering the status of the car to a different value other than *"emergency"* or *"out of service"*. In both cases, firing of transition *Restart* results in a car's token being returned to place *Cars* and place *Database* being updated; therefore, the car is accessible by other sub-models.

The transition stage describes the process of moving elevator cars between floors. This is modeled by the firing of transition *Transfer*, which is enabled if transition *Maintain* is not, the car's desired-floors list is not empty, and the car's current floor matches no calls of the desired-floors list. After firing transition *Transfer*, the car's token is updated as follows. If the car's desired-floor list has calls beyond the car's current floor, it continues shifting in the same direction. Otherwise its direction is reversed. In both cases, the token in place *Database* is updated accordingly.

Once a car reached its desired destination, it is in the arrival stage. At this stage, transition *Arrive* is enabled if transition *Maintain* is disabled and the car's current floor matches a requested call of the desired-floors list. After firing transition *Arrive*, car's token is updated by dropping the requested floor from the car's desired-floor list. Additionally, the car's state is set to one of three cases. If the car's desired-floor list has more calls, then it continues serving the requested calls. Otherwise, the car is set to idle if the car's current floor agrees with its parking floor or alternatively the car is dispatched to its parking floor with an appropriate direction. Finally, place *Doors* represents doors' operations. Since such operations are almost trivial, they have been included in one place that can be converted into an hierarchical sub-model to show all of the doors' activities.

4.5 The Parking Optimizer Sub-Models

Holding idle cars on or near floors where most hall-calls are placed substantially improves passenger satisfaction and the system's energy usage and efficiency [15]. Therefore, cars are initially distributed in fair distances between the lowest and highest floors. Subsequently, the parking optimizer sub-models continuously analyse the placed hall-calls and then assign the elected floors to the cars. The parking optimizer models include the election sub-model

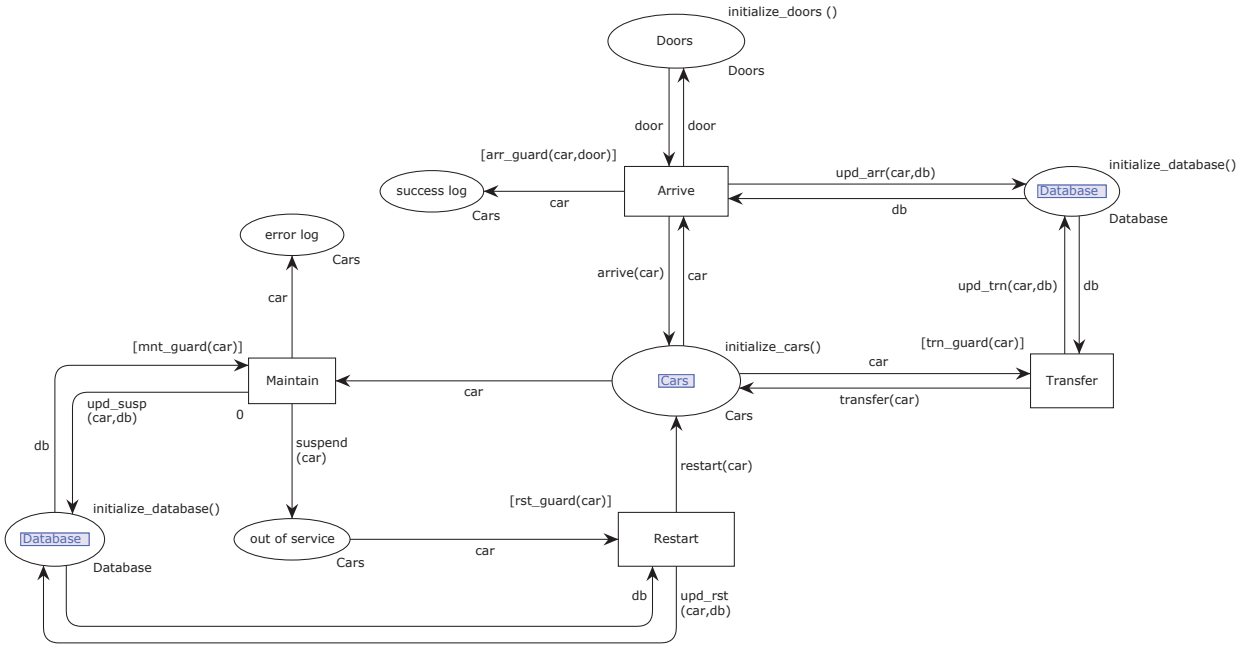


Fig. 4: The System-Cycle Sub-Model

and the position sub-model. In Table 9, the definitions of the parameters which facilitate the control of the parking optimizer model are presented.

Table 8: The Parameters of The Parking Optimizer Sub-Models

Parameters	Legal values
Parking system state	$\{"enable", "disable"\}$
Analyzing hall calls	$\{ x \mid x \in \mathbb{Z} \}$

All models of the parking optimizer sub-model are self-explanatory. However, it is also important to first note that the procedures of all models must be sequential. Therefore, some transitions have priorities which are represented by numbers that appear in the bottom-left corner of each transition. The value of these numbers implies the order of enabled transitions. Second, the fusion place *Lock of the system (Lock Sys)* is functionally similar to an inhibitor arc. If there is a token in a place, an inhibitor arc disables a transition (see [16]). For example, if *Lock Sys* has the value "0", then transition *Count Call* is disabled: this locks the system from analysing more hall calls. This place is critically important because when a car is in the maintenance stage of the system-cycle sub-model and not accessible, then the parking optimizer sub-models cannot successfully assign all elected floors. Consequently, the parking optimizer sub-models are suspended until the ongoing maintenance is completed.

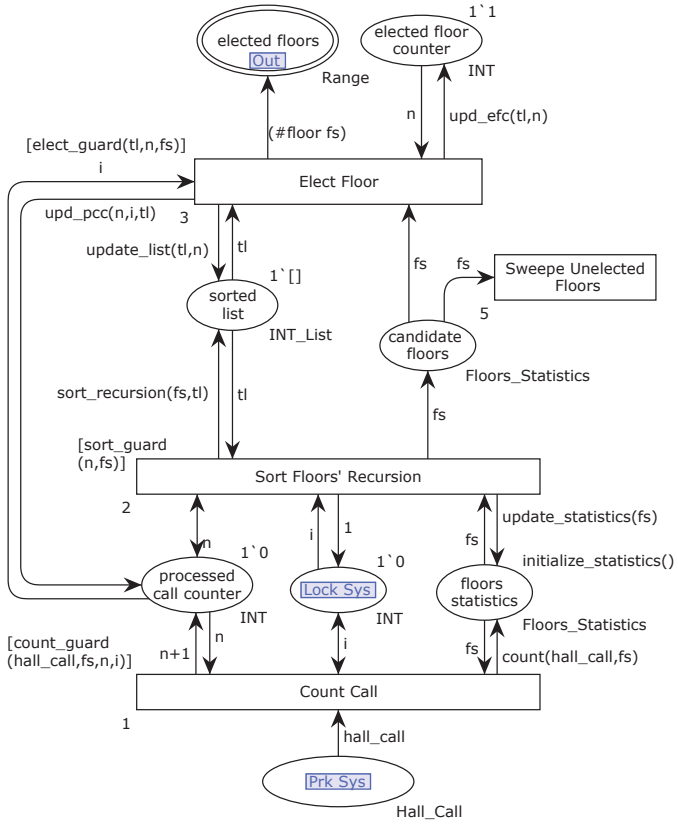
After election sub-model (in Figure 5(a)) counts the repetition of all placed hall-calls and then nominates the floors where most hall-calls have been repeatedly placed, the position sub-model (in Figure 5(b)) alters the cars' parking floors with respect to their scopes. This process works to approximately guarantee fair distances between cars to reduce the total wait times.

Table 9: The Colour Sets of The Parking Optimizer Sub-Models

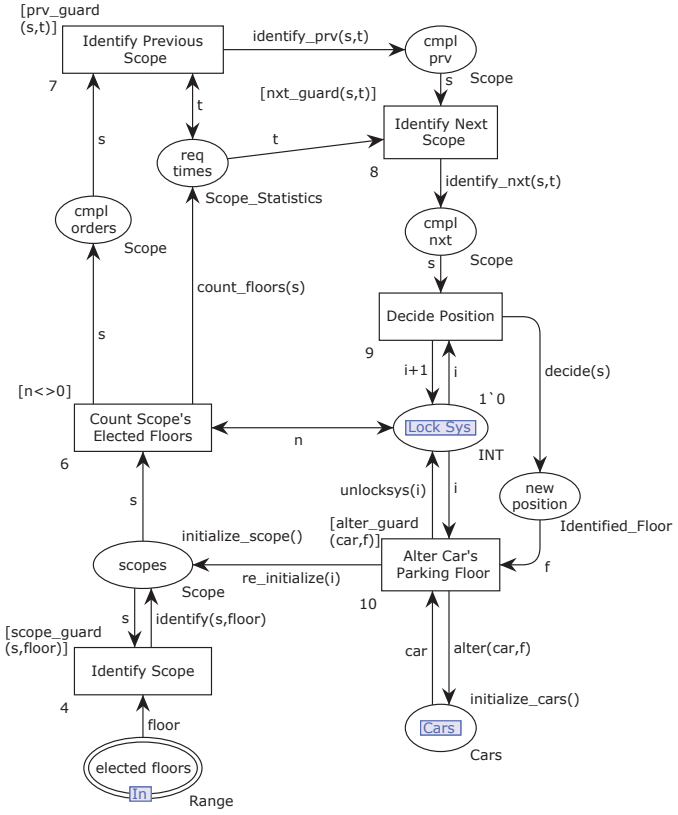
Colour Sets	Definitions
Floors Statistics	$\{$*(floor,times)* $\mid$ *floor* $\in$ *Range, reputation* $\in \mathbb{Z}\}$
Scope	$\{$*(scope id,prev,next,elected floors)* $\mid$ *scope id* $\in$ *Car ID, prev* $\in \mathbb{Z}$, *next* $\in \mathbb{Z}$, *elected floors* $\in$ *INT List*$\}$
Scope Statistics	$\{$*(scope id,floors' number)* $\mid$ *scope id* $\in$ *Car ID, floors' number* $\in \mathbb{Z}\}$
Identified Floor	$\{$*(floor id,floor's number)* $\mid$ *floor id* $\in$ *Car ID, floor's number* $\in$ *Range*$\}$

5. Analysis

Two analyses techniques were applied. The first technique is the reachability analysis by means of the State Space tool [17]. This tool verified and generated an automatic report. The proposed model has dead markings that occur in cases such as a placed hall-call with no available car. Transition *Maintain* and transition *Restart* are dead transitions which indicate no operation failure of the proposed model. The second technique is the simulation-based analysis by means of CPN Tools. Although this technique is flexible, it is also time-consuming. However, the proposed model was



(a) The Election Sub-Model



(b) The Position Sub-Model

Fig. 5: The Parking Optimizer Sub-Models

simulated repeatedly with different settings, and the entire definition of the system was achievable.

6. Conclusion

We have provided a fairly general CPN-based model of the elevator system. The model covers various aspects of the elevator system and is divided into five sub-models that can be analyzed independently. Such division allows for easier tracking of errors and faults in the elevator system. The flexibility of the model allows for easy adaptation of different algorithms and rules depending on the actual needs.

Acknowledgements

The first author was supported by Prince Sattam bin Abdulaziz University, the second author was supported by the Ministry of Education of Saudi Arabia, while the third author acknowledges partial support by NSERC Discovery Grant of Canada.

References

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Eds., *Fundamentals of Software Engineering*, 2nd ed. Pearson Prentice Hall, 2003.

[2] C.-H. Lin and L.-C. Fu, "Petri net based dynamic scheduling of an elevator system," in *IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 192–199.

[3] Y.-H. Huang and L.-C. Fu, "Dynamic scheduling of elevator systems over hybrid Petri net/rule modeling," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1998, pp. 1805–1810.

[4] Y. C. Cho, Z. Gagov, and W.-H. Kwon, "Timed Petri net based approach for elevator group controls," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 1999, pp. 1265–1270.

[5] E. S. Etessami and G. S. Hura, "Abstract Petri net based approach to problem solving in real time applications," *Fourth IEEE Region 10 International Conference*, pp. 234–239, 1989.

[6] Y. Ahmad, Farooq and Fakhir, Ilyas and Khan, SherAfzal and Khan, "Petri net-based modeling and control of the multi-elevator systems," in *Neural Computing and Applications*, vol. 24. Springer London, 2014, pp. 1601–1612.

[7] J. M. Fernandes, J. Baek Jorgensen, and S. Tjell, "Requirements Engineering for Reactive Systems: Coloured Petri Nets for an Elevator Controller," in *14th Asia-Pacific Software Engineering Conference*, 2007, pp. 294–301.

[8] D. Liqian, Z. Qun, and W. Lijian, "Modeling and analysis of elevator system based on timed-coloured Petri net," in *Fifth World Congress on Intelligent Control and Automation*, vol. 1, 2004, pp. 226–230.

[9] J. Ye, J. Li, F. Deng, and C. Wang, "Simulation of the intelligent control circuit based on Petri net," in *6th International Conference on Computer Science & Education*, 2011, pp. 66–69.

[10] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," in *Proceedings of the IEEE 77.1*, 1989, pp. 81–98.

[11] K. Jensen, "Coloured Petri Nets and the Invariant Method," *Theoretical Computer Science*, vol. 14, no. 3, pp. 317–336, 1981.

[12] K. Jensen, *Coloured Petri Nets*. Springer, 1994.

[13] CPN Tools AIS Group, The University of Technology, Eindhoven, The Netherlands, http://www.cpntools.org.

[14] K. Jensen and L. M. Kristensen, "Coloured Petri Nets Modelling and Validation of Concurrent Systems." Berlin: Springer, 2009.

[15] G. Barney, *Elevator Traffic Handbook: Theory and Practice*. Taylor & Francis, 2003.

[16] R. Janicki and M. Koutny, "Semantics of inhibitor nets," *Information and Computation*, vol. 123, no. 1, pp. 1–16, 1995.

[17] K. Jensen, S. Christensen, and L. M. Kristensen, "CPN Tools State Space Manual," *Department of Computer Science, Univerisity of Aarhus*, 2006.

Semantic Approach for Traceability Link Recovery using Uniform Resource Identifier (STURI)

Mazen Alobaidi, Khalid Mahmood
Department of Computer Science and Engineering
Oakland University,Rochester, USA
Email: malobaid@oakland.edu,mahmood@oakland.edu

***Abstract*—The efficiency and effectiveness of traceability link recovery in requirements management is becoming increasingly important within Requirement Engineering due to the complexity of project developments, such as continuous change in requirements, geographically dispersed project teams, and the complexity of managing the elements of a project - time, money, scope and people. Therefore, the traceability links among the requirements artifacts, which fulfill business objectives, is so critical to reducing the risk and ensuring the success of products. To that end, this paper proposes a semantic based traceability link recovery (STURI) architecture that presents the meaning of texts in Uniform Resource Identifier derived from Linked Data. To the best of our knowledge, this is the first architectural approach that uses Uniform Resource Identifier for finding similarities among requirements and among the automation of the recovery traceability.**

***Keywords*-Semantic Web, NLP, Linked Open Data, Document Similarity, Uniform Resource Identifier**

I. Introduction

The success of the project relies on robust requirement management tools. Requirement management is the most effective phase of project development. As a requirements management expert, Peter Zielczynski defines the major steps in requirements management as follows: establishing a plan, eliciting requirements, developing the vision document, creating use cases, supplementary specification, and finally a system design. Indeed, the backbone of requirements management is "Requirements Traceability" by which it is possible to map individual artifacts of requirements with other artifacts in the system. In addition, requirements traceability identifies and outlines the lineage of each requirement, apart from its backward traceability (derivation), its forward traceability (allocation) and its association to other requirements. Traceability, according to the IEEE Standard Computer Dictionary [1], is defined as the degree to which a relationship can be established between two or more artifacts (document) of the development process. Traceability is employed to guarantee solution conformance to requirements and to assist in scope and change management, risk management, time management, cost management, and communication management. Furthermore, it is used to distinguish missing functionality or to identify if implemented functionality is not supported by a specific requirement.

In current approaches, the relationship among requirement engineering artifacts is represented by their Semantic Similarity and Semantic Relatedness. Semantic Similarity is usually defined by considering the lexical relations of synonymy, or equivalent words and hyponymy, or the type-of relation. Semantic Relatedness, on the other hand, extends the definition of similarity by examining all types of semantic relations that connect two concepts. Such relations include, in addition to the above-mentioned two similarity relations, the antonym, metonymy, or the relations between wholes and parts, functional relations of frequent association as well as other non-classical relations. Although, enormous progress has been made in traceability link recovery using the current approaches, our proposed approach is more unique from the perspective of sharing information, this has been achieved by making use of a unique and uniform mechanism of identification, the Uniform Resource Identifier (URI) [3]. Our approach can be summed up in the twofold principle: considers all terms/words in a document as a resource and each resource uniquely identifies web resources.

The recovery/generation of the links of the traceability matrix, which utilizes URI among the artifacts, will be highly accurate as all nouns are compared with respect to URI rather than a key-word, synonymy, hyponymy, wholes and parts relation. One of the motivations of this project is to develop an automated traceability matrix for project managers using a Semantic based Traceability Link Recovery (STURI) framework that will use Uniform Resource Identifier, the Linked Open Data (LOD) [7], and Natural Language Processing [10] to find association among various artifacts of projects.

The paper is structured as follows. The next section is the related works. Section 3 contains the preliminary. In section 4, the concept, architecture and design of STURI are presented. The evaluation and results are shown and discussed in section 5, and the paper is concluded in section 6.

II. Related Works

There are two distinct disciplines of research that are associated to our proposed approach, namely Information Retrieval technique (IR) [11]–[13], and Semantic technique [25]–[27].

A. Information Retrieval Technique

Recovery traceability links has been extensively studied in Information Retrieval (IR). IR is mainly defined as discovering processes of nominee traceability links on the basis of the similarity between software engineer artifacts that can be transforming in some unstructured test format [14]. Antoniol, Giuliano et al. [14] proposed a method based on Information Retrieval (IR) to establish and maintain traceability links between source code and free text document where documents are ranked against queries constructed from the identifiers of source code. Ali et al. [16] proposed an approach to establish and maintain traceability links between source code, software requirements, and software repository requirements as well as improve the precision and recall of information retrieval (IR) techniques by showing that mining software repositories (MSR) and combining the mined outcomes with IR techniques improves the precision and recall of requirement traceability links. Marcus et al. [17] proposed approaches that advocates for the use of latent semantic indexing (LSI) to recover traceability links between documentation and source code. . All the above proposed approaches based on IR, however, lack accuracy (precision, recall) [16], perform poorly in short context [18], and disregard word order, syntactic relations, morphology, semantic relation, and word ambiguities [19].

B. Semantic Technique

Semantic Similarity and Semantic Relatedness have recently received the attention of researchers who are studying traceability link recovery. Semantic Similarity is usually defined by considering the lexical relations of synonymy, or equivalent words, and hyponymy, or the type-of relation [20]–[22], [26], [27]. Semantic Relatedness, on the other hand, extends the definition of similarity by examining all types of semantic relations that connects two concepts [23]–[25]. Zhang, Witte et al. [19] proposed an approach which creates traceability links between source code and documentation software that can be summarized as follows: building ontologies, modeling the domains of source code and software documents, creating a knowledge base by automatically population these ontologies through code analysis and text mining, and finally establishing traceability links between code and documents through ontology alignment. Falbo et al. [28] contributed to this semantic approach by proposing an extended semantic document management platform for the requirement domain by using semantic annotations in requirement documents. Furthermore, there is an exploration of the conceptualization established by the proposed software requirements, with reference to its ontology and the generation of the traceability matrix, both of which are based on a dependency relationship and related axioms (reasons). Mahmoud et al. [29] also offered another appoint, based on semantic relatedness, which brings human judgment to an earlier stage of the tracing process by integrating it into the underlying retrieval mechanism. This would use a measure based on Wikipedia, namely Explicit Semantic Analysis. Although all the mentioned approaches improve accuracy, they have some limitations: only one database is used, which recovers traceability between source codes and software artifacts only, which in turn ignores word ambiguities.

III. Preliminary

A. Linked Data

The Web enables us to link related documents. Similarly it enables us to link related data. The term Linked Data refers to a set of best practices for publishing and connecting structured data on the Web. Key technologies that support Linked Data are URIs (a generic means to identify entities or concepts in the world), HTTP (a simple yet universal mechanism for retrieving resources, or descriptions of resources), and RDF (a generic graph-based data model with which to structure and link data that describes things in the world) [8].

B. Linking Open Data

Linking Open Data (LOD) project is to extend the Web with a data commons by publishing various open datasets as RDF on the Web and by setting RDF links between data items from different data sources.

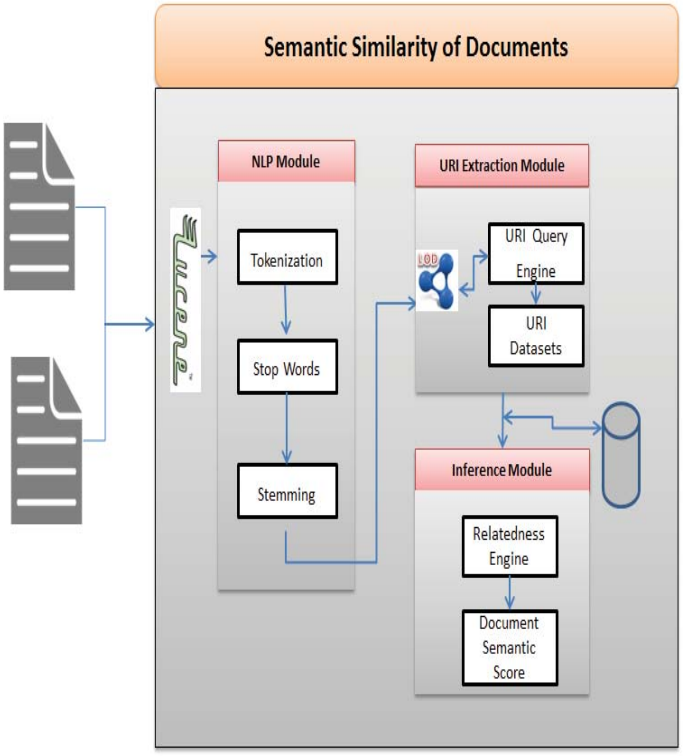


Fig. 1. Semantic based Tractability Link Recovery Architecture (STURI)

IV. Semantic based Tractability Link Recovery Architecture (STURI)

Due to the complexity of the product development within diverse industries, it is imperative that the recovery of traceability links in requirements management be efficient and effective. We propose a solution that derives document similarities from the Uniform Resource Identifier contained in the

compared Documents. Figure 1 shows the system architecture of components. Our architecture is comprised of three components. Namely: "Natural Language Processing (NLP)", "URI extraction" , and "Inference Engine". The following sections present a detailed description of each of the above components. Since semantic similarity between Documents is used URI semantic similarity, we will first describe our method for measuring URI semantic similarity.

A. URI semantic similarity

Approaches for measuring semantic similarity have been developed in the previous decade. Different similarity methods have proven to be useful in some specific applications: namely, artificial intelligence, natural language processing, information retrieval, and data mining. we proposed a document similarity measure which provides the greatest correspondence to common sense. Given two documents, document a and document b, we need to find the semantic similarity s(a,b). We can do this by implementing and evaluating using LOD as the underlying reference ontology, we get all URIs for each word in documents 'a' and 'b' and we measure the similarity based on our weighting scheme and overlapping URIs.

B. NLP module

The NLP module uses Apache Lucene framework [4] for uploading the artifacts, tokenization, stop words, stemming, and lemmatization [10]. Tokenization is the process that splits the artifacts into tokens, stop words are words which are filtered out, where stemming and lemmatization are the process of converting or removing the inflexional, derivational form to a common world form.

C. Uniform Resource Identifier Extraction module

The important tasks of this module are extracting RDF statements from LOD, building RDF store, building SPARQL query, executing query, URI disambiguation, and creating URI result map collocation. The process of this module is as follow:

1) Extraction RDF triples and building RDF graph store: to extract triples from LOD we download Dataset and read all RDF triples of the dataset. Also, we created RDF graph store using Jena tools.
2) Building SPARQL query: to find all the URIs of a resource, we dynamically construct SPARQL query that match all subjects of RDF triple with input resource term.
3) URI Disambiguation: We use the Naive Bayes classification technique [31] to classify the document and the dereferenced URI resource. In turn, we conducted a string match between classified classes of that document and resource; if there is a match, then we consider the URI as a candidate
4) Creating URI result Map: to improve the performance of Inference engine, we constructing map collection to hold all the URIs of document.

Algorithm 1 shows the pseudo code of our implementation

Algorithm 1 URI Extraction module Pseudo code

```
1: procedure GET-TRIPLES (file)
2:     dataset ← createDataSet(file)
3:     model ← dataset.getDefualtModel()
4:     return model
5: end procedure
6: procedure GET-URISET (Word, model)
7:     Querystring ← sprqlFactory.creat(Word)
8:     Query ← QueryFactory.creat(Querystring)
9:     result ← Query.execute(Querystring, model)
10:    return result
11: end procedure
```

D. Inference Module

The Inference module functionality maintains a count for the Overlapping of URI between two artifacts and the calculation of a "similarity score". In addition, it generates candidates for traceability links. Figure 2 shows the URI semantic algorithm data structure. This module provides the following:

1) Reading all URIs of each term in the text
2) Building two space dimension matrix that represents terms in URIs
3) Creating graph models represent all terms that shares same URI
4) Recording the length of the path between nodes in a graph. The path represents the distance between two terms
5) Calculating the average length path for each graph
6) Creating URImap collection based on the average length path greater than or equal to URIThreshold
7) Using the URImap to measure the relatedness between two texts by calculating the URI overlapping

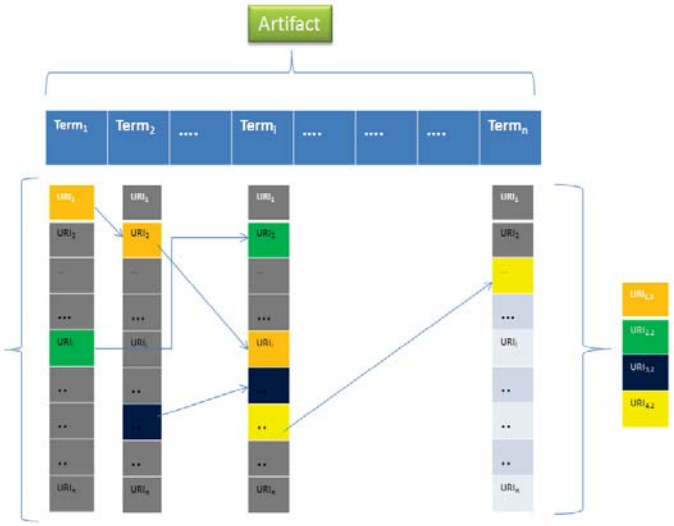


Fig. 2. URI Semantic Algorithm Structure

After the URI sets of two documents are formulated, Semantic similarity between documents can be calculated by

$$\mathrm{Sim}(\mathrm{URIMAP_a}, \mathrm{URIMAP_b}) = \frac{\mathrm{URIMAP_a} \bigcup \mathrm{URIMAP_b}}{\mathrm{URIMAP_a} \bigcap \mathrm{URIMAP_b}} \quad (1)$$

V. Evaluation

In order to measure the efficiency and the accuracy of the proposed framework, we implemented an experimental framework using General Architecture and Engineering Text Lucene and Jena which are Apache development environments that provide a rich set of interactive tools for the creation, measurement and maintenance of software components for processing human language. Ultimately, in order to simulate a real world scenario, We have selected three different datasets.

A. DataSets

The datasets used are illustrated below and see table I.

1) MODIS: The NASA Moderate Resolution Spectrometer (MODIS) dataset [30] is a small dataset created from the full specification (high- and low-level requirements documents) for the MODIS space instrument software. This dataset contains 19 high-level requirements, 49 low-level requirements, and a validated true positive 41 links that we refer to as the "True traces".

2) CM-1: The dataset consists of a complete requirement and a complete design document for a NASA space instrument [30]. The dataset contains 235 high level and 220 low-level requirements. The trace for the dataset was manually verified. The "theoretical true trace" (answerset) built for this dataset consisted of 361 correct links. Each of the high and low-leve files contain the text of one requirement element.

3) Standard Company: A dataset of requirements management tools from Borland CaliberRM. The dataset contains 27 high level and 27 low-level requirements. The "theoretical true trace" (answerset) built for this dataset consisted of 19 correct links.

TABLE I
DataSets

DataSet	High Req	Low Req	Traces
ModisDataset	19	48	39
CM-1	235	220	361
Standard Company	27	27	19

B. Experimental Results

In the experiment, the accuracy of our approach framework is compared against Vector Space Model [11] and Wu Palmer algorithm [27]. The primary accuracy measures used for comparison are Precision and Recall. Precision is the ratio of the number of true positive links retrieved over the total number of links retrieved where Recall is the ratio of the number of true positive links retrieved over the total number of true positive links.

We have implemented VSM, Wu Palmer and STURI algorithms as part of a requirement tracing tools that we have built called "Dynamic Trace Workbench". We used the datasets mentioned above. To evaluate STURI approach against VSM and Wu Palmer, we have run two experiments.

1) First Experiment: We ran each algorithm mentioned above using all datasets. The results of the running experiments were collected and analyzed against existing answer sets and the results information were used to calculate Precision and Recall for evaluation. Figure 3,4, and 5 compare the Recall measurement achieved by STURI and those for two benchmark techniques. The analysis of the result shows that STURI provides better Recall measurement cross all datasets. On the other hand, Figure 6, 7, and 8 shows that STURI preforms better than Wu Palmer algorithm in Precision, however VSM gives us better precision.

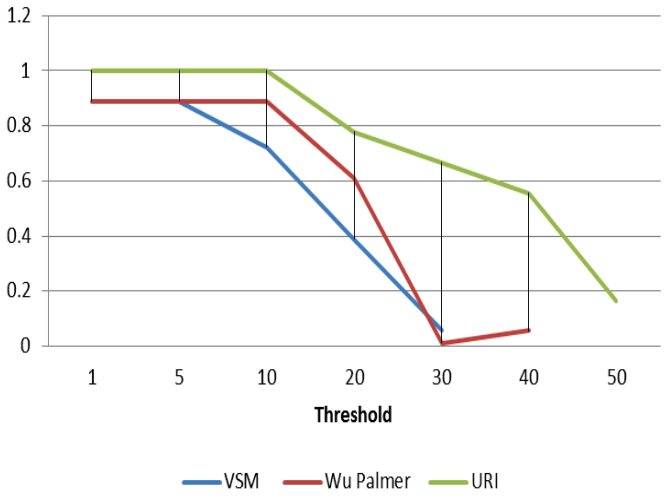


Fig. 3. Recall for Standard Company Dataset

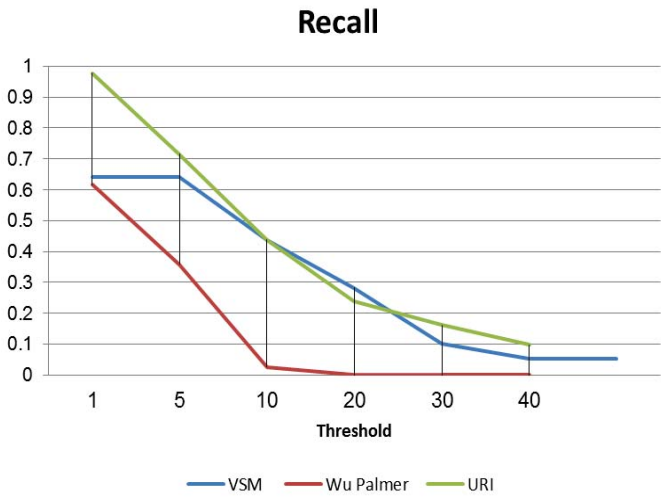


Fig. 4. Recall for Modis Dataset

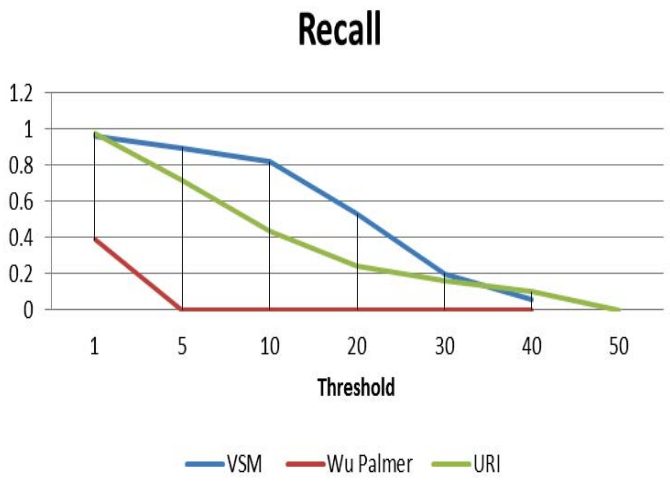


Fig. 5. Recall for CM1 Dataset

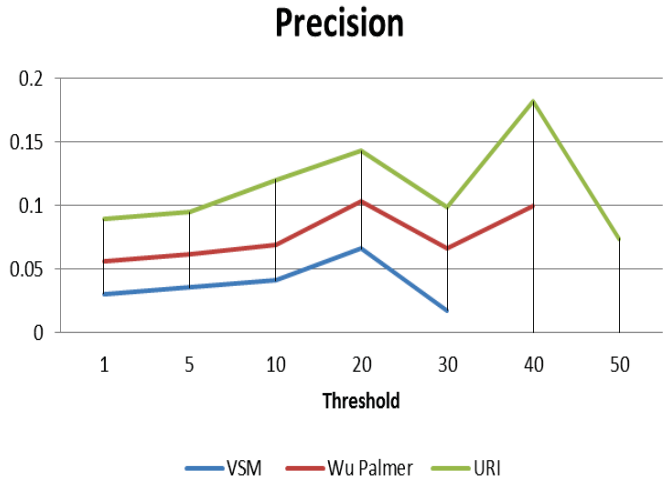


Fig. 6. Precision for Standard Company Dataset

2) Second Experiment: The second experiment began by using the VSM algorithm against one of the datasets from above. We ran the STURI against the same dataset, and then we aggregated the two results and calculated the Recall and Precision. Moreover, we repeated the same experiment against the other two sets, as well as with Wu Palmer and STURI. The analysis of the result shows, on average, a correspondence of 30 percent on Recall. Recall improved on combining VSM/WU and STURI over running VSM or Wu Palmer alone. In addition, the results show on average a Precision improvement of 3 percent when combining VSM/WU and STURI over running VSM or Wu Palmer alone. In this experiment we presented a subset of the results, see Figure 9 and 10.

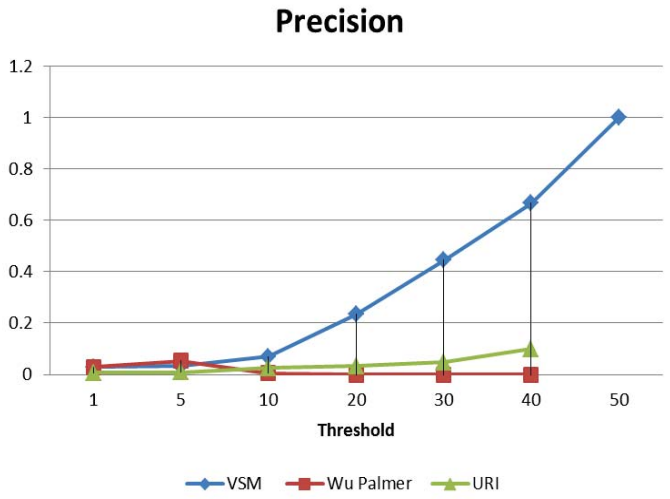


Fig. 7. Precision for Modis Dataset

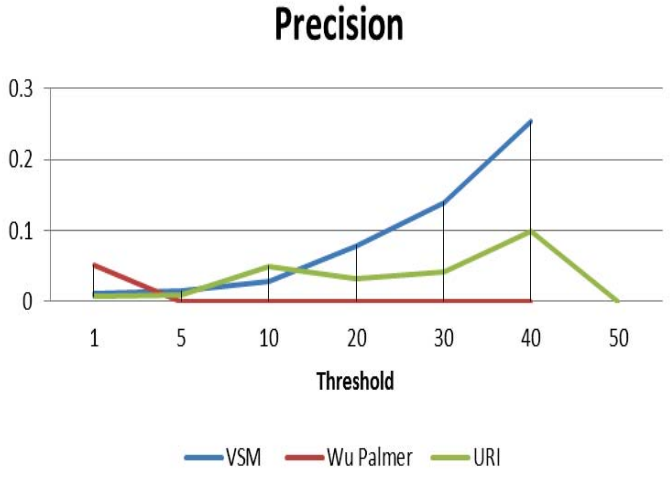


Fig. 8. Precision for CM1 Dataset

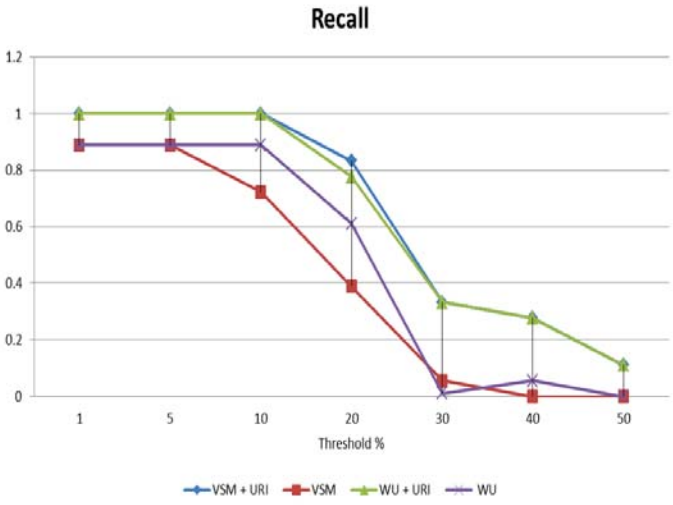


Fig. 9. Recall for Standard Company Dataset on combine approach

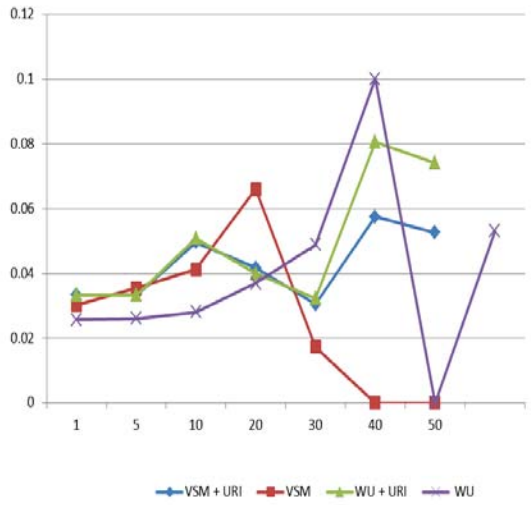


Fig. 10. Precision for Standard Company Dataset on combine approach

VI. Conclusion

These inter-industrial projects are constantly updated and modified in light of new risks and developing products. Traceability links are a vital part of requirements managements for these industries. Since Automated Traceability Links are an important element to ensure the success of Software engineering projects, our proposed framework helps Software engineer projects to meet business requirements, to improve the precision and recall of traceability links between requirements artifacts, and increases the efficiency of time management.

References

[1] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991.

[2] wiki.dbpedia.org : About. [Online]. Available: http://dbpedia.org/About

[3] "RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax." [Online]. Available: https://tools.ietf.org/html/rfc3986

[4] "Apache Lucene - Welcome to Apache Lucene." [Online]. Available: http://lucene.apache.org/

[5] "Freebase API - google developers." [Online]. Available: https://developers.google.com/freebase/

[6] "OpenCyc for the semantic web." [Online]. Available: http://sw.opencyc.org/

[7] "SweoIG/TaskForces/CommunityProjects/." [Online]. Available: http://www.w3.org/wiki/SweoIG/TaskForces/

[8] "Linked Data | Linked Data - Connect Distributed Data across the Web." [Online]. Available: http://linkeddata.org/home

[9] BabelNet 2.5 - a very large multilingual encyclopedic dictionary and semantic network. [Online]. Available: http://babelnet.org/

[10] The stanford NLP (natural language processing) group. [Online]. Available: http://nlp.stanford.edu/

[11] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[12] V. V. Raghavan and S. M. Wong, "A critical analysis of vector space model for information retrieval," *Journal of the American Society for information Science*, vol. 37, no. 5, pp. 279–287, 1986.

[13] D. L. Lee, H. Chuang, and K. Seamons, "Document ranking and the vector-space model," *Software, IEEE*, vol. 14, no. 2, pp. 67–75, 1997.

[14] A. Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. Binkley, "Evaluating test-to-code traceability recovery methods through controlled experiments," *Journal of Software: Evolution and Process*, vol. 25, no. 11, pp. 1167–1191, 2013.

[15] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *Software Engineering, IEEE Transactions on*, vol. 28, no. 10, pp. 970–983, 2002.

[16] N. Ali, Y. Gueneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *Software Engineering, IEEE Transactions on*, vol. 39, no. 5, pp. 725–741, 2013.

[17] A. Marcus, J. I. Maletic, and A. Sergeyev, "Recovery of traceability links between software documentation and source code," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 05, pp. 811–836, 2005.

[18] D. Metzler, S. Dumais, and C. Meek, *Similarity measures for short segments of text*. Springer, 2007.

[19] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological approach for the semantic recovery of traceability links between software artefacts," *IET software*, vol. 2, no. 3, pp. 185–203, 2008.

[20] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," *arXiv preprint cmp-lg/9511007*, 1995.

[21] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," *arXiv preprint cmp-lg/9709008*, 1997.

[22] P. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," *arXiv preprint arXiv:1105.5444*, 2011.

[23] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet:: Similarity: measuring the relatedness of concepts," in *Demonstration Papers at HLT-NAACL 2004*. Association for Computational Linguistics, 2004, pp. 38–41.

[24] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis." in *IJCAI*, vol. 7, 2007, pp. 1606–1611.

[25] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.

[26] R. Rada, H. Mili, E. Bicknell, and M. Blettner, "Development and application of a metric on semantic nets," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 1, pp. 17–30, 1989.

[27] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1994, pp. 133–138.

[28] R. de Almeida Falbo, C. E. C. Braga, and B. N. Machado, "Semantic documentation in requirements engineering."

[29] A. Mahmoud, N. Niu, and S. Xu, "A semantic relatedness approach for traceability link recovery," in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*. IEEE, 2012, pp. 183–192.

[30] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[31] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.

A new Framework for Quality-Based SOA Migration

Ayman Massoud

Computer Science, Otto von Guericke University, Magdeburg, Germany

Abstract *– Several software migration frameworks are presented to support the planning and the implementation activities needed to migrate the legacy software components to a new software paradigm like SOA "Service-Oriented Architecture". However, the migration process has a quality challenge that required more research attention. Legacy to SOA migration process is required to deploy and consider some sort of quality evaluation and measurements to make sure that the migrated systems will be able to conduct reliable and efficient business results in any given services transaction. This paper is introduced to provide a new proposal framework for quality-based SOA migration that adopted measured qualified phases of service identification, target architecture design, service implementation, deployment, and evaluation.*

Keywords: SOA Migration Framework, SOA Migration Measurements, SOA Migration Quality Requirements

1 Introduction

Many organizations are still relying on complex legacy systems to automate their business practices and collect, process, and analyze its business data. These systems are heterogeneous, distributed, constantly evolving, dynamic, long-lived, and mission critical that presented as a backbone of the enterprise operations. To optimize business value, there is a need to modernize these systems using a new software paradigm like SOA. SOA migration process enables the organization to benefits from the new service-orientation capabilities, making the legacy functionalities more robust, efficient and cost effective to align easily with the new business opportunities.

Despite the fact that the SOA migration process is succeeded to make the legacy systems running under modern paradigm and derived benefits from its new features, there are some of legacy limitations and problems are still exist, and some of the migration outcomes are not efficient as expected. Therefore, SOA migration process should execute under qualified approach that consider the quality characteristics in all its migration phases. This paper presented to discuss how to design, implement, and evaluate new quality-based SOA-migration framework that improve the quality level of the migration products.

2 Legacy to SOA Evolution: A Systematic Literature Review

We conducted a systematic literature review to collect legacy to SOA evolution approaches (Selected paper from our previous work [1]) reported from 2005 to 2014 (publications with high citation count) such as:

- Architecture-Driven Modernization - ADM [2]
- IBM's SOMA Method [3], [4]
- Service Migration and Reuse Technique - SMART [5] , [6]
- SOA Migration Framework SOA-MF [7], [8]
- SOA Migration - SOAMIG [9]
- Consolidation framework of structural legacy to SOA Migration [10]
- Advanced Software based-service provisioning and migration of legacy Software [11]

And found that the most presented legacy migration frameworks are considered deeply technical analysis of understanding the legacy system and the transition steps to the target system. However, considering the quality requirements and measurements throughout the migration tasks still need more research contributions to avoid repeating of legacy issues and limitations in the new environment, and to produce more reliable, integrity, and efficiently SOA solution.

We conduct a comparison between several selected approaches on four subjects (Migration Phases, Legacy Paradigm Change, Migration Goals, and the Quality Considerations and Evaluation Measurements) [1], due to the limitation of paper size, we describe a brief comparison example Table 1:

Table 1: Examples of SOA Migration Methods

Method	Quality Considerations and Evaluation, Measurements
ADM Method [2]	- Not defined. - Refer to software assurance and metrics that should to be adopted during transformation processes.
SOMA Method [3], [4]	- Provides support of monitoring and management the business processes and performance in the production environment. - Provides linkages to runtime management aspects.

SMART Method [5], [6]	Considered concrete analysis of the migration feasibility, risk, and involved cost.
SOAMIG Method [9]	- Not defined. - Used testing and migration cut-over tools.
ARTIST Project [11]	- The evaluation measurements are not considered, but the ARTIST consider quality check and V&V certifications model to make sure that the project deliverables are match good level of migration quality.

3 Proposal Model for Migration Quality Requirements

Building qualified migration framework is the main goal of this research, which required to consider specific quality characteristics and controls. In this section we provide a new proposal model (Figure 1: Quality Requirements Model in SOA Migration) represents the major quality requirements that needed to deal with the quality challenges during the SOA migration processes. This model of quality requirements is based on three migration directions (Target System Design, Migration Implementation (Process Integrity), and Evaluation and Measurements).

This model' scope is determined based on the most migration phases that affected in the solution outcomes efficiency.

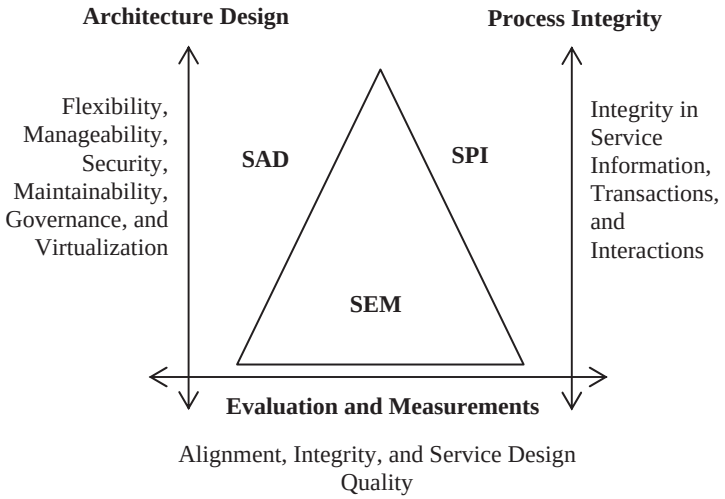


Figure 1: Quality Requirements Model in SOA Migration

The mentioned selected publications and the state of the art of transforming the legacy systems to SOA processes are displayed that there are different frameworks varying from high level abstraction of migration phases up to re-engineering processes that targeting legacy architecture modernization, including model-driven based approach, reverse/forward engineering methods, SOMA, SMART, SOA-MF, SOAMIG, and others migration architectures. We concluded that the most critical quality directions that formulate the quality level in SOA-Migration process could classified into three topics:

- SOA Architecture Design
- SOA Process Integrity
- SOA Evaluation and Measurements

3.1 SOA Architecture Design- SAD

The quality requirements in target system planning and design phase are intend to choose the architecture design and its related SOA technologies, which eventually plays an important role in the efficiency and adaptability of the future SOA system. Basically, target system understanding can be viewed from two perspectives: functional characteristics and technical characteristics:

- The functional characteristics include the potential functionalities to-be evolved from the legacy code. This process is referred to service design and application composition. It also defines to what level of granularity the services are to be defined and, accordingly, the orchestration of the services has to be managed to support business processes. Various functional and non-functional properties should also be considered, such as maintainability, interoperability, responsiveness, performance, security, and availability.

- The technical characteristics of the target environment include service technology (SOAP or REST-based), messaging technologies, communication protocols, service description languages, and service discovery mechanisms.

The proposed model figure 1 (Quality Requirements Model in SOA Migration) is considered six major characteristics that shape the power of SOA architecture design, including Flexibility, Manageability, Security, Maintainability, Governance, and Virtualization.

3.2 SOA Process Integrity – SPI

SOA process integrity is the ability to conduct reliable business activity in a consistent SOA environment with seamless integration at every interacted and participated service. In general, process integrity is the critical component of SOA implementation, the ability to synchronize between services, human tasks, information, applications, domains and users in a secure, scalable SOA environment. Business must be agile enough to deliver the same reliability, consistency and predictability in an open service-oriented system as in a tightly coupled closed system. In SOA, the role of migration/integration is not only to bridge the islands legacy systems, but also to deal with the process integrity/consistency issues. Process integrity has three main elements:

Transaction integrity: Ensures that individual updates of business and IT resources are linked and processed as a single unit of work, all completing successfully or being rolled back in case of technical or business failure.
Interaction integrity: Ensures that elements of people interactions with business and IT systems are interact and remembered wherever and whenever those interactions occur in secure, scalable, and reliable environment.
Information integrity: Helps deliver trusted, secured information to business processes, regardless of delivery channel, operational platform (IT or people), and information lineage, in which the information to be meaningful, accurate, correctness, and aligned.
So, the quality requirements model recommended to apply some sort of integrity mechanisms to avoid the pitfalls that could be encountered when extending SOA infrastructure from limited-scope projects to a broader enterprise wide implementation, and describes how the considering of the integration quality can help to deliver on the promises of service-orientation approach [12].

3.3 SOA Evaluation Measurements - SEM

After converting legacy systems to be services by transformation the legacy code (migration approach) or by exposing/interfacing the legacy functionalities (integration approach), these services have to be deployed. Some necessary activities are required to manage and control the behavior of services during usage**.** Monitoring the service behavior is very important to maintain the service performance, validation, integrity, etc… Service controlling has been a research challenge in the SOA domain due to the dynamic uses of the services in the SOA context. Build business logic using the legacy services is needed to be controlled to validate the integration process workflow, services input/output, and services data mapping. Another important topic is service quality measurements, measuring the services description, security, data consistency, and others measurements that support the services quality. The mentioned quality model is considered these kinds of research issues by providing several considerations during the design phase, and provides integration evaluation metrics to measure and evaluate the evolved services.

4 SOA Migration Framework - SMF

SMF is presented as a new proposal of SOA migration framework that considers the quality requirements as an essential need throughout all the migration activities, and adopted the E4 approach [13] of the process measurements to evaluate the migration processes.

As shown in figure 2, **SMF** includes five major migration phases that applied the E4 measurement approach (see point 5 SMF and E4 Evaluation measurements).

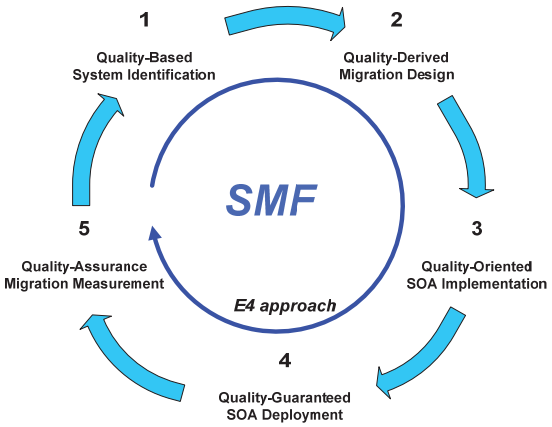


Figure 2: Conceptual SOA Migration Framework – SMF

The following sub-sections will describe the SMF phases in brief; we discuss design roles, activities, tasks, artifacts, and guidance concerning tasks. Collectively this constitutes the SMF method. Figure 3 illustrates a typical process flow of an engagement executing the SMF method in high abstraction level, including 21 major activities that represented sequentially the flow of SMF method. Each activity may produce one or more artifact products, the relation between the mentioned quality model and the migration activities artifacts will be displayed in subsection 4.6.

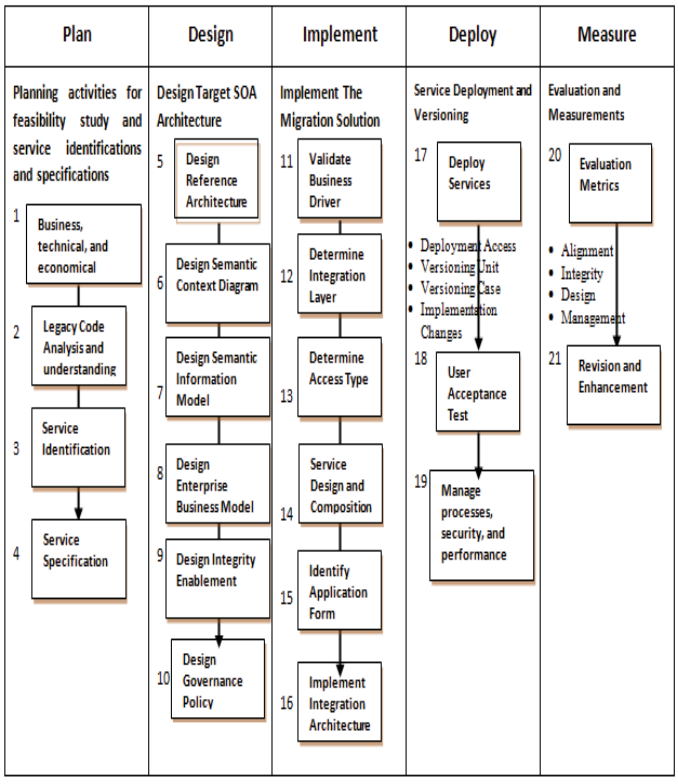


Figure 3: SMF Life-cycle high-level flow

4.1 Quality-Based SOA System Identification

System identification phase is presented as a migration planning phase, interested in four elements (Feasibility Study, Legacy Code Analysis, Service Identification, and Service Specification) that addresses the issues of making the migration feasibility study, this phase is aim to decide if the existing legacy systems are needed and ready to be migrated to SOA solution from the technical and business perspectives, discuss which technical methodology and approach is a proper one used to understand the existing legacy code and its component's structures and functionalities, and also this phase is concerning in how to identify the candidate part of the legacy code to be re-presented as a reusable service in the target SOA architecture.

4.2 Quality-Derived SOA Migration Design

Design phase is aim to understand the SOA key principles, architecture, and environment. Define the main SOA components to be designed, and which technology, standards to be used. Also, in this phase some issues like performance, security, governance, integrity, and others SOA characteristics to be discussed. Design phase support to facilitate the representation of the desired SOA architecture, enables the design of the target architecture with major components of the SOA environment, standards to be used, quality of service (QoS) expectations, and interaction patterns between services.

In **SMF** the design phase is considered that the architecture design should align between the legacy systems characteristics and the enterprise business models toward efficient migration process. So, to achieve this objective, SMF provides the required architecture tools for the design components including SOA Reference Architecture, Enterprise Semantic Context and Information models, Enterprise Business Process Model, Integrity Enablement's, and Governance Controls.

4.3 Quality-Oriented SOA Implementation

Several techniques are presented to implement the migration process. SMF adopted the wrapping technique (fastest, less risky and cost effective technique) to migrate the legacy systems by interfacing it to other software via web services. It is a black-box modernization technique, since it focuses on the interface of the legacy systems, hiding the complexity of its logic. Also, the re-engineering technique is target to add the SOA capabilities and functionalities to the existing legacy systems via reverse engineering, and redesigning the existing software.

SMF is adopted the integration strategy to migrate to SOA architecture, and use the mix between the re-engineering and wrapping strategies to implement the services needed to build the migration solution. Integration enables disparate resources to share business data. **SMF** provides its implementation approach in the following steps:

- Validate the migration business drivers
- Determine which architectural layer to perform the integration activities
- Identify the implementation access type
- Designing Service Implementation
- Identify the integration application form
- Implement the integration architecture

4.4 Quality-Guaranteed SOA Deployment

After implemented the necessary services which exposing the candidate legacy functionalities, the exposed services are then deployed in the service infrastructure and tested to determine if the expected functionalities are formed and integrated correctly. A successful deployment is require a service provisioning that includes activities such as publishing and discovering services in a repository, maintaining Quality of Services (QoS), versioning, testing, and evolution of services that lead to the proper functioning of the services and ensure that the SOA environment operates reliably and efficiently.

SMF considered in the guaranteed the deployment and versioning phase by allowing service implementations to evolve without breaking existing consumers, leading to more services loosely coupled, minimize the impact of versioning, and reduce the amount of deployed code. In SOA, service versioning considered the coexistence of multiple versions of the same service, which allows each consumer to use the target version that it is designed and tested for. In this multiple coexisting versions of the same service, the system allows for the independent life cycles of services and their consumers and minimizes the overall impact of changes to new version.

4.5 Quality-Assurance SOA Measurements

Having deployed services is not enough to move the existing legacy enterprise systems from the islands platforms to SOA environment. SMF is considered that in order to complete the migration project efficiently and successfully, there is a need to a right kind of service, well designed and properly built service, efficient service communication, and reliable service that is able to satisfy the current and the future business requirements. Proposal SMF migration framework is focuses on how we can improve the quality factors on SOA-Migration.

SMF describes the migration process:

$$\overset{\text{System\_1/N}}{\underset{\text{Legacy Architecture}}{}} \boldsymbol{ARC} \Rightarrow \overset{\text{System\_1/N}}{\underset{\text{SOSE}}{}} \boldsymbol{ARC} \qquad (1)$$

Where **ARC** is refer to Software Architecture

and also describes the migration metrics and measurement as follows:

$$\begin{matrix}\text{quality\_improvements}\\ \text{scheduled usage}\end{matrix} MP \begin{matrix}\text{Legacy Systems}\\ \text{research\_approach}\end{matrix} \xrightarrow{\textit{quality\_criteria}}$$
$$\begin{matrix}\text{quality\_improvements}\\ \text{scheduled usage}\end{matrix} MP \begin{matrix}\text{SOSE}\\ \text{research\_approach}\end{matrix} \quad (2)$$

Where **MP** is refer to Measurement Process

Also, SMF considered the quality improvements can be described as follows:

- Efficiency Measurements = {cost, *performance, flexibility*}
- Consistency Measurements = {Data Validation, *Service Interactions , Service Transactions*}
- Level of Service-Interoperability = {Input Validation, Output Validation}
- Level of Loose-Coupling = {Independent Services, Dependent Services}
- Characteristics of Island Systems = {Overlapping Object, Limited Function, Semantic dissonance, Inconsistent Data, Insufficient Business Workflow, Lack of Enterprise Data and Business Model}

Evaluating the quality of the provided SMF solution is divided into two steps:
- Measure the goals and the objectives achieved from the migration process, in other words, have the legacy issues been recovered after migration or not?
- Evaluate the overall SMF solution from several business and technical perspectives.

4.6 SMF Quality Relationships

SMF designed its quality level based on 3 bases:

1- The quality of the migration approach that built on the business-driven concept, and to maximize the ROI from the migration process.

2- The quality of each migration activity which presented by the artifacts produced to support it, these artifacts are aligned with the proposed quality model, such as (Template, Model, Reference Design, Assessment Guidance, Recommendation, Best Practices, Techniques, and Metrics), for more details see Table 2 that describes some of SMF Artifacts.

3- Evaluating the migration phases by using the E4 approach which adopted the concept of Establish, Extract, Evaluate, and Execute. To control the risk and to determine the areas of improvement (will described in the next section).

Table 2: Examples of SMF Artifact Products

SMF Artifact	Description and Function
Legacy Issues Template	List of the common legacy systems issues that gathering from academy and industry experiences, understand how the existing legacy issues affected the current business operations which support to determine the level of the business criticality.
Migration Risk Assessment	List of risk assessment questioners from business and technical perspectives, this assessment support to understand and identifying the challenges and its mitigations and rollback methods, determine the resources and existing capabilities which support the migration decision.
SMF Quality Evaluation	Considering five items of quality evaluation including validation, integrity, interoperability, loose coupling, and island characteristics. This evaluation is represents the most important quality aspects that support the SMF efficiency.
LCA Model	Model of legacy code and system understanding, including Reverse Engineering and Visualization Quality Check, Delta Analysis, and Documentation Understanding.
SMF-RA Reference Architecture [14]	SMF reference architecture is facilitates services and design communications and provides a representation of progress and evolution of the legacy to SOA solution in high-level abstraction diagram. **SMF-RA** represents the logical design of the legacy to SOA solution, provides architecture layers that represent the separation of concern, and the relations between the architecture blocks, and used as a blueprint that supports the project stakeholders using templates and guidelines during the migration and the solution development life-cycle.
Integration Efficiency Considerations [15]	Provide efficiency considerations and recommendations to design the integration architecture, including Messaging Infrastructure, Message Broker, Web Services, Web Services Wrappers, Direct Database and Adapters Access, and ESB.
SMF Services Evaluating (Metrics Table)	Metrics table to evaluate and measure the service functionality, quality, and efficiency. This assessment will guide to understand the maturity level of the migration services throughout the migration phases, and put spots on the area of improvements and issues.

5 SMF-E4 Evaluation Measurements

As mentioned in the previous point, evaluating and measuring each migration activity is one of the most essential quality requirement that recommended by the SMF quality model. For this purpose, we adopted the E4-Measurement Process, The E4–measurement process (Figure 4) consists of four essential steps: **Establish** concrete objectives and the measurement and analysis scope and activities. **Extract** measurements for the established need. **Evaluate** this information in view of a specific background of actual status and goals. **Execute** a decision to reduce the differences between actual status and goal.

Figure 4: The E4 measurement process [13]

The following figure 5, represented that each **SMF** activity applied the E4 measurement approach to serve and comply with the required quality aspects enabled by the migration quality model (figure 1). And then produced one or more artifact products to support the migration tasks as follows:

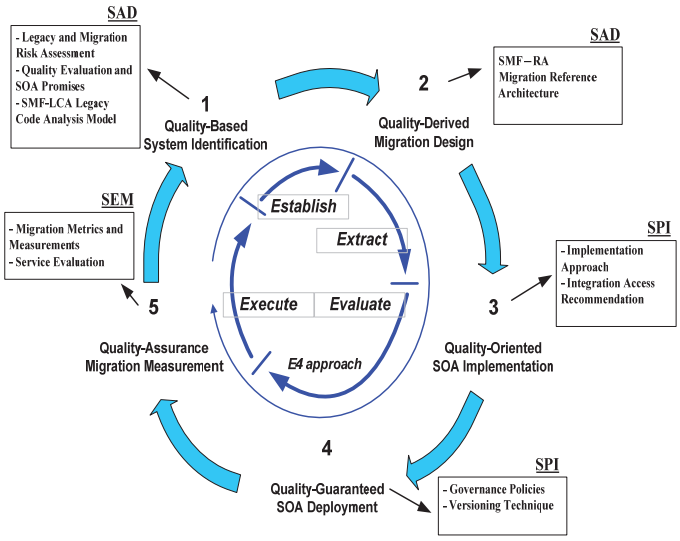


Figure 4: SMF - Major Artifact Products

The following Table 3, we provide an example illustrated in general form how the E4 approach can be applied in SMF first stage of qualified-based system identification:

Table 3: SMF and E4 Application

SOA Migration Framework SMF and E4 Measurements Process System Identification Phase (General Approach)
➢**Establish**: the measurements intentions are based on the consideration of the legacy systems limitations and challenges, and its expected goals from the new SOA paradigm in aligning with the enterprise business needs. • Strategy: draw the best possible software migration process • Concept: choose a class of software systems as business applications • Development: keep the requirements of the quality improvement • Evolution: determine the appropriateness of the migrated systems in the new paradigm
➢**Extract**: extract the right information that supports the established measurements, and extracted using a goal-driven method. • Strategy: analyze the current and the estimated architecture • Concept: determine the risk analysis and the expected ROI benefits • Development: understanding legacy functionalities • Evolution: determine the potential services
➢**Evaluate**: evaluate the legacy and the new paradigm characteristics. • Strategy: evaluating the migration decision • Concept: determine the metrics values of the legacy and the SOA systems • Development: align with the business requirements • Evolution: determine the migration approach
➢**Execute**: make the decision and implement the start implement the change. • Strategy: choose the business unit for migration • Concept: select the appropriate tools and resources • Development: implement the desired services • Evolution: keep the development in standards and high quality level

6 Conclusion

To improve the quality level of the SOA migrated systems, we concluded that there is a need to apply a qualified software migration framework that considered the quality requirements in its migration activities. In this paper, we present 5-phase and 21 major activities of qualified SOA migration framework SMF which applied the E4 measurements process to measure and evaluate the migration stages and its sub-processes.

We provide the following contributions:

- Design new SOA Migration Framework-SMF of quality-based SOA migration, that consists of five phases: quality-based system identification and architecture design, quality-oriented SOA implementation, quality-guaranteed SOA deployment, and quality-assurance migration measurements.

- Provide new model of SOA migration quality requirements, and applied it in the SMF framework.

And adopted the process measurements E4 in the new SMF approach, which support to measure and improve the quality level of the migration activities.

- Design and implement several artifacts products that support the process of quality improvement such as (Template, Model, Reference Design, Assessment Guidance, Recommendation, Best Practices, Techniques, and Evaluation Metrics).

7 References

[1] Massoud, A.: "Quality-based Issues in SOA Migration"; Software Measurements News"; Journal of the Software Metrics Community. Vol. No. 20, Issue No. 1, Page 19-52, Feb 2015.

[2] Khusidman, V.; Ulrich, W.: "Architecture-Driven Modernization. Transforming the enterprise"; OMG. Draft Vol. No. 5, 2007.

[3] Arsanjani, A.; Ghosh, S.; Allam, A.; Abdollah, T.; Ganapathy, S.; Holley, K.: "SOMA: A method for developing service-oriented solutions"; IBM Sys. J. Vol. 47, Issue No. 3, Page 377–396, 2008.

[4] Fuhr, A.; Horn, T.; Riediger, V.; Winter, A.: "Model-driven software migration into service-oriented architectures"; CSRD. Vol. No. 28, Issue No. 1, Page 65–84, 2011.

[5] Lewis, G.; Morris, E.; O'Brien, L..; Smith, D.; Wrage, L.: "SMART: The service-oriented migration and reuse technique"; CMU/SEI, Tech. Rep. CMU/TN-029, Sep 2005.

[6] Lewis, G.; Smith, D.: "Service-oriented architecture and its implications for software maintenance and evolution"; FoSM'08 IEEE. Page 1–10, 2008.

[7] Razavian, M.; Lago, P.: "A frame of reference for SOA migration. Towards a Service-Based Internet"; Springer. Page 150–162, 2010.

[8] Razavian, M.; Lago, P.: "A survey of SOA migration in industry. Service-Oriented Computing"; Springer. Page 618–626, 2011.

[9] Zillmann, C.; Winter, A.; Herget, A.; Teppe, W.; Theurer, M.; Fuhr, A.; Horn, T.; Riediger, V.; Erdmenger, U.; Kaiser et al.,U.: "The SOAMIG Process Model in Industrial Applications"; CMSR'11, IEEE. Page 339–342, 2011.

[10] Khadka, R.; Saeidi, A.; Jansen, S.; Hage, J.: "A Structured Legacy to SOA Migration Process and its Evaluation in Practice"; Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA). IEEE 7th International Symposium, Page 2-11, Sep 2013.

[11] ARTIST Project: Advanced Software-based Service Provisioning Migration of legacy Software, ARTIST Newsletter, http://www.artist-project.eu, Apr 2015.

[12] Massoud, A." Process Integrity in SOA Migration"; In: Büren et al.: Praxis der Software-Messung. Shaker-Verlag, Aachen, S. Page 205-222, 2014.

[13] Ebert, C.; Dumke, R. "Software Measurement – Establish, Extract, Evaluate, Execute"; Springer. 2007.

[14] Massoud, A.; Dumke, R.: "Efficient Reference Architecture for Integrated Legacy Applications based SOA"; In: Abran et al.: IWSM/Mensura Proceedings 2012. Assisi, Italy, CPS Publishing Service of IEEE, Session 1B, 2012.

[15] Massoud, A.; Dumke, R.: "Efficient SOA-based Integration of Legacy Applications"; In: Schmietendorf/Patzer: BSOA 2012 -7. Workshop Bewertungsaspekte serviceorientierter Architekuren, Shaker-Verlag, S. Page 95-104, 2012.

A Directive-Based Transformation Approach for UML Class Diagrams

Devon M. Simmonds
Department of Computer Science
University of North Carolina, Wilmington
Wilmington, North Carolina, 28403
simmondsd@uncw.edu

Abstract

In a model driven engineering (MDE) environment, developers create and evolve applications by creating models and transforming abstract models to more concrete models. Model transformation languages are needed to realize the benefits of MDE. In this paper we describe the Directive-Based Transformation Approach (DBTA), an imperative transformation approach for lightweight transformations on class models. DBTA utilizes abstractions that are specific to class diagram transformations and helps remove some of the accidental complexities associated with more general purpose transformation languages such as QVT. DBTA uses schemas consisting of transformation directives that can be processed by an interpreter. We present the overall design of DBTA and illustrate the new approach using Fowler's extract super class refactoring transformation.

Keywords: Class diagrams, model transformations, transformation schemas, transformation directives, middleware, CORBA.

1. Introduction

A model transformation is a process that takes as in-put one or more source models and produces one or more target models [1 - 9]. The MOF 2.0 Query View Transformation (QVT) Language [16] is an Object Management Group (OMG) standard for specifying model transformations. Our experience specifying transformations [18 - 21] using QVT suggests that transformation languages need to work at a high level of abstraction in order to reduce the accidental complexity associated with specifying non-trivial transformations. QVT includes an operational mappings language, a core language and a relations language. The relations language defines a transformation as a set of relations between source models and target models, where a relation consists of a source domain pattern that describes valid source models and a target domain pattern that describes valid target models. Each pattern is an object diagram consisting of instances of metamodel classes. Several problems arise when models are described in terms of metamodel class instances. First, these specifications can produce large descriptions. Second, expressing transformations at this level of granularity can be tedious for medium to large-sized models. For example, a QVT source pattern that describes class models consisting of two classes with one attribute each, and one association between the classes, will contain instances for the classes, the attributes, the attribute types, the association and the association ends, that is, at least 9 model elements. Third, it is not easy to differentiate between types of model elements if they are all represented as instances of metamodel classes.

We propose the Directive-Based Transformation Approach (DBTA) [19], a lightweight graphical domain specific transformation approach for UML [22] class diagrams. DBTA provides a transformations specification syntax based on the concrete syntax of UML class diagrams rather than on the abstract syntax (i.e., the class diagram metamodel). Class diagrams are one of the most commonly used diagram types in object-oriented modeling and thus, one can expect that class diagram transformations will have wide applicability.

DBTA is designed to: (1) make transformation specifications more understandable to individuals familiar with UML class diagrams, (2) support the development of mechanisms for representing model transformations at a more abstract level than object diagrams, and (3) explore the use of directives in expressing model transformations. As a lightweight specification, DBTA is not intended to be a replacement for QVT. Instead, the intent of the research is to explore approaches and techniques that can enhance the development of model transformation standards such as QVT.

In this paper we describe the design of DBTA and illustrate use of the new approach. We provide a description of the transformation process and transformation directives in Section 2 and illustrate the use of DBTA for specifying the extract super class transformation [15] in section 3. Discussion and conclusions are presented in section 4.

2. The Directive-Based Approach

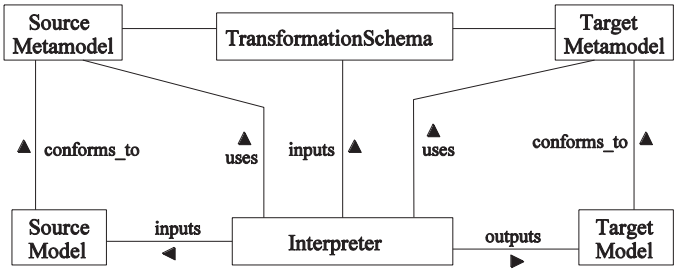


Figure 1. Model Transformation Concepts.

Figure 1 shows a class diagram representation of MDE transformation concepts in DBTA. The *Source Model* represents the model that is being transformed while the *Target Model* represents the transformed class model. The source model conforms to the *Source Metamodel* while the target model conforms to the *Target Metamodel*. The notion of conformance used in this paper is that established by Kim et al. [11]. In general, a model A conforms to a model B, if A faithfully reflects the structural and behavioral constraints and properties defined in B.

DBTA defines transformations graphically using a class model transformation specification called a *Transformation Schema*. Transformation schemas contain imperative statements called *transformation directives* that stipulate how target model elements are formed from source elements. Transformation directives are processed by an *Interpreter* to perform the specified transformations on class models.

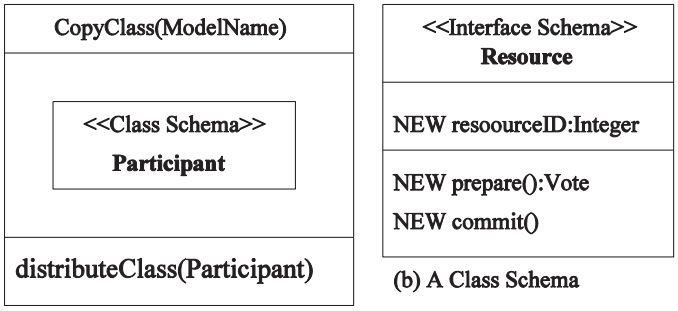


Figure 2. Transformation Schema Elements

2.1. Overview of Transformation Schemas

To reflect the class diagram syntax, a transformation schema is divided into three compartments. The first compartment contains the name of the transformation, formal arguments or parameters to the transformation, and any preconditions on the transformation. Figure 2 (a) shows the structure of a transformation schema. In the figure, the first compartment has the name *CopyClass(ModelName)*, where *CopyClass* is the name of the transformation and *ModelName* is the formal argument to the transformation. Formal arguments to a transformation represent the UML class models on which the transformation will be performed.

The second compartment of a transformation schema contains constructs called element schemas. Element schemas contain directives for creating target class model elements from source models elements. There are different types of element schemas corresponding to model elements of UML class diagrams, for example, class schemas, interface schemas, operation schemas and association schemas. Class schemas may contain attribute schemas and operation schemas. To accommodate this, class and interface schemas are also divided into compartments. A class schema is divided into a *Name Directive* compartment, an *Attribute Directive* compartment, and an *Operation Directive* compartment. Attribute schemas are specified in Attribute Directive compartments and operation schemas are specified on Operation Directive compartments.

For example, Figure 2 (b) shows an interface schema with the name *Resource*, containing the *NEW prepare():Vote* and the *NEW commit()* operation schemas in its Operation Directive compartment and the *NEW resourceID:Integer* attribute schema in its Attribute Directive compartment. In these examples, the keyword *NEW* is an example of a transformation directive. The *NEW* directive is used to create new class model elements. We describe directives in section 2.3.

The third compartment of a transformation schema contains a list of transformations to be performed after the transformation defined in the first two compartments. In Figure 2, the *distributeClass* transformation will be executed after the *CopyClass* transformation has been executed. The use of transformation schema compartments allow the modularization of transformations and the specification of a transformation `program' by listing modularized transformations in the second and third compartments of a transformation schema.

2.2. Transformation Process

DBTA can be used to support the transformation process shown in Figure 3 [18, 20.21]. The figure has two activity partitions. The *Development of Transformation* partition shows behavior associated with creating transformations and the *Application of Transformation* partition shows behavior associated with the use of transformations.

A complete DBTA class model transformation specification consists of a *Source Pattern* that describes valid source models and a transformation schema. The Source Pattern and transformation schema are created during the Develop Model Transformations activity. We describe model patterns using RBML templates, a variant of the Role Based

Meta-modeling Language (RBML) [11 - 13]. RBML is a UML based language that supports rigorous specification of pattern solutions, where a pattern solution characterizes a family of solutions for a recurring design problem. RBML class diagram templates have template model elements that are explicitly marked using the "|" symbol (see Figure 4). Each tem-plate model element represents a model element in the source model. The template model elements in the class diagram template must be replaced by the source model elements they represent before a diagram template is used. In essence, we use RBML model patterns to specify the metamodel of source models.

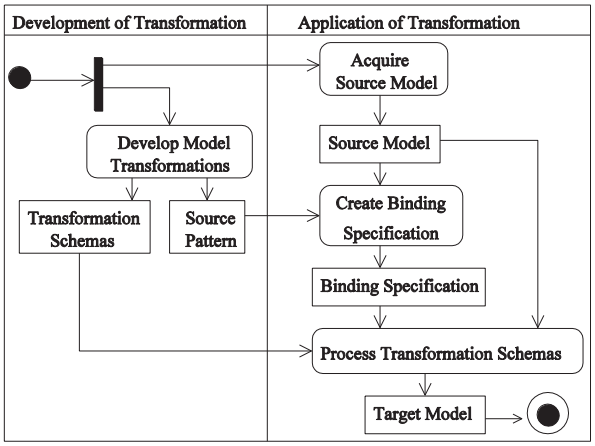


Figure 3. Model Transformation Process [18,20,21].

The Acquire Source Model activity results in the development or acquisition (e.g., from a model repository) of the source model. A source model element that is to be transformed is represented in a transformation directive by its corresponding source pattern model element marked using the "|" symbol. Therefore, the source model element that each source pattern element represents must be identified, before directives in a transformation schema can be executed. This identification is done using a *Binding Specification*, which is a listing of source pattern model elements and corresponding source model elements. The binding specification is developed during the *Create Binding Specification* activity.

During the *Process Transformation Schemas* activity, the directives in the transformation schema are executed, resulting in the target models. The inputs to this activity are the Source model, the Binding Specification and the Transformation Schemas.

2.3. Transformation Directives

Transformation directives are imperative statements that stipulate how target model elements are formed. Five directives are used in transformation schemas. These directives are: *source*, *rename*, *exclude*, *new* and *redefine*. Constraints on models may also be specified using a *when* statement. In the examples presented in this paper, transformation directives in the figures, are written using uppercase letters and the first and last compartments of the transformation schemas are not shown.

The *source* Directive: The *source* directive is used to select source model elements for inclusion in the target model. When the selected model element is to be modified, additional transformation directives are required. When the source directive is the only directive associated with a model element, the model element is copied to the target model without modification.

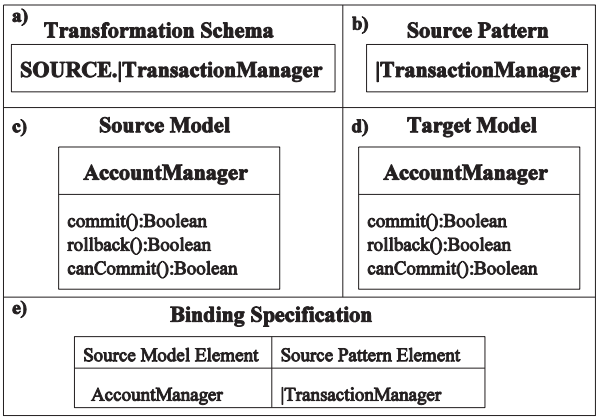


Figure 4. Example of the source directive.

The use of the source directive is illustrated in Figure 4. In the figure, the transformation schema has the single source directive: source.|TransactionManager, where |TransactionManager is a source pattern model element that represents a source model class that manages a transaction (e.g., the 2-phase commit protocol). The meaning of this directive is that the class in the source model that |Transaction Manager represents should be copied to the target model. This source model class is shown in Figure 4 (c). Before the transformation can be effected, the source model class is identified using the binding specification shown in Figure 4 (e). From this binding specification, |TransactionManager is bound to the AccountManager (the actual transaction manager class). This source directive results in the AccountManager class and its operations being copied to the target model. When a model element is copied, all its properties and any constraints associated with the model element are also copied. The source directive has the following forms.

1. *source*.Parent[RenameDirective].
2. *source*.Parent.SubElement [RenameDirective].
3. source.Parent.Property[.SubElement].MetaAttribute.

Parent is a reference to a composite source pattern model element and *SubElement* is a reference to an operation,

operation template, attribute or attribute tem-plate, defined in the model element bound to *Parent*. *RenameDirective* is an optional *rename* directive, *property* is a keyword, and *MetaAttribute* is a meta-attribute of *Parent* or *SubElement*.

The *rename* Directive: Consider the scenario in which the *AccountManager* class from the previous example is being copied to an environment in which the name *ServiceManager* is used for transaction managers. The name of the copied class must be changed to reflect this environmental requirement. The *rename* directive is used to effect these kinds of transformations. The rename directive is used to provide a context-specific name for a model element. The rename directive has the form:

ModelElement {name = modelElementName}

where *ModelElement* is a reference to a model element in the source pattern and *modelElementName* is the context-specific name to be given to the source model element bound to *ModelElement*.

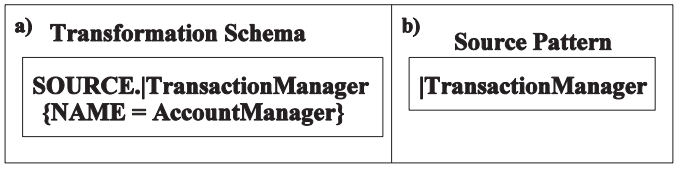


Figure 5. Modifying a class using the *rename* directive.

The rename directive is illustrated in Figure 5. The figure shows a class schema with the rename directive: {*name=AccountManager*}, attached to the source directive. For the source model shown in Figure 4 (c), the source.|TransactionManager directive results in the AccountManager class being copied to the target model, and the rename directive {*name=AccountManager*}, results in the name of the copied AccountManager class being changed to *ServiceManager*.

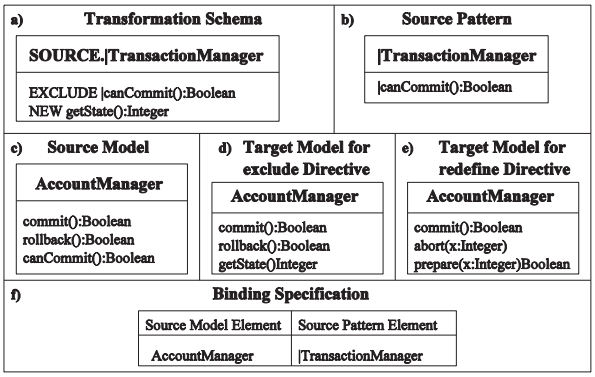


Figure 6. Examples of the *exclude* and *new* directives.

The *exclude* and *new* Directives: The exclude directive is used to exclude source model elements from inclusion in the target model. For example, if the AccountManager input class must be copied to the target model but its *canCommit* operation must be excluded, we can specify an exclude directive as shown in Figure 6 (a). The effect of the directive is to eliminate the canCommit operation from the target model as shown in 6 (d). The exclude directive may be applied to any model element. The exclude directive has two forms:

1. exclude ModelElement,
2. exclude

where the directive is associated with a transformation schema association or other transformation schema relationship (e.g., dependency and generalization) that is to be excluded from the target model.

The new directive is used to specify a new model element or a value for a meta-attribute. For example in Figure 6(a), a new operation is added to the target model using the directive, *new getState():Integer*. The new directive has three forms:

1. [new] ModelElement,

   where *ModelElement* is the specification of a new class diagram element, for example a new class or a new operation.

2. [new]

   In this form, the directive is associated with a transformation schema association or transformation schema relationship that is to be created.

3. [new] *Property*.metaAttribute = newValue

The *redefine* Directive: Consider the scenario in which the *AccountManager* input class in Figure 6 is being copied to a Jini [10] middleware environment where the following requirements typically hold:

1) The name *abort* is used for the *rollback* operation.
2) The abort operation has an integer parameter but does not return a value.
3) The name *prepare* is used for *canCommit*.
4) The prepare operation has an integer parameter.

These requirements may be realized using the *redefine* directive. Using the *redefine* directive, modifications are specified using `rename`, `new` and `exclude` directives. In this example, the redefine directive:

redefine |rollback{name=abort}(new id:Integer, exclude <params>):exclude Boolean

transforms the *rollback* operation by:

a) Changing the name of the operation to *abort* using the {name=abort} rename directive.
b) Adding a new integer parameter using the *new id:Integer* directive.
c) Causing all other parameters of the source model operation to be deleted using the *exclude <params>* directive.

Similarly, the directive:

redefine |canCommit {name=prepare}(new id:Integer, exclude <params>):Boolean

transforms the canCommit operation.

2.4. The *when* Statement

Preconditions are specified for transformations and directives using a *when* statement. Preconditions on transformations are described in the first compartment of a transformation schema after the name of the transformation. Preconditions on directives are specified immediately after a directive. The *when* statement has the forms:

(1) transformation-name *WHEN* expression
(2) directive *WHEN* expression

In this statement, *transformation-name* is the name of a transformation, *directive* is a transformation directive and *expression* is a logical expression. The *when* statement is illustrated in Figure 7.

3. Refactoring Example: Extract Superclass

Model refactoring is an important class of transformations in model driven engineering. In model refactoring, the structure of a model is transformed without changing the behavioral properties of the model [15]. Models are typically refactored to improved one or more model properties. For example, a model may be refactored to enhance reusability of specific model components or to make a model amenable to distribution.

This section illustrates the representation of the extract superclass model refactoring transformation de-scribed by Fowler [15]. In the extract superclass transformation, class operations with the same (or similar) operation signature are extracted from source classes and used to form a super class.

A transformation schema that describes the extract superclass model transformation is illustrated in Figure 7. The precondition on the transformation is that an operation with the same signature must exist in two different classes. In such cases, a new super class is created and populated with matching operations as well as matching class attributes. Preconditions are specified for transformations and directives using a *when* statement. Preconditions on transformations are described in the first compartment of a transformation schema after the name of the transformation. Preconditions on directives are specified immediately after a directive.

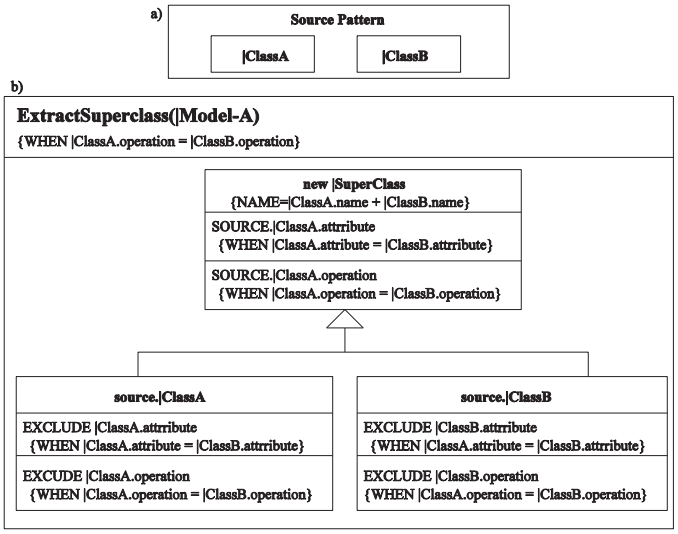


Figure 7: Extract Superclass Transformation Schema

A source model for the transformation is shown in Figure 8 (a). The source model has four classes, Employee, Manager, Company, and Customer. The execution of the transformation proceeds by binding one of the four classes to |ClassA, and then binding the other three classes to |ClassB, one at a time to determine if the precondition may be satisfied.

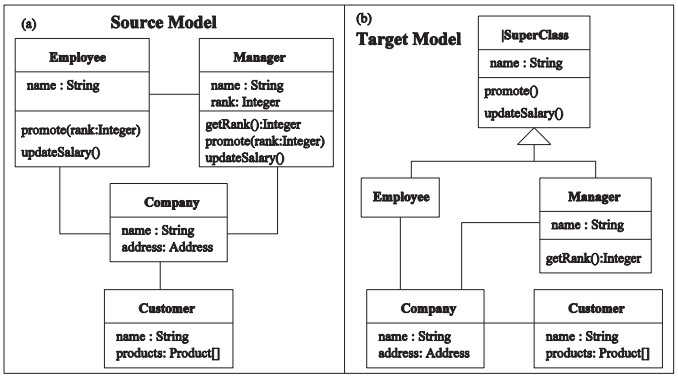


Figure 8: Source and Target Models for Extract Superclass Transformation

If |ClassA is matched to the Employee class, then binding the Company and Customer classes to |ClassB does not satisfy the precondition because these two classes have no operation

in common with the Employee class. However, when the Manager class is bound to |ClassB, then |ClassA.operation in the precondition match the *promote* and *updateSalary* operations in the Employee class and |ClassB.operation in the precondition match the *promote* and *updateSalary* operations in the Manager class. Since the precondition is satisfied, the transformation is executed and a new super class is created with copies of these two operations. In addition, the *name:String* attribute is common to both Employee and Manager, so this attribute is copied to the super class as well. The resulting target model is shown in Figure 8 (b).

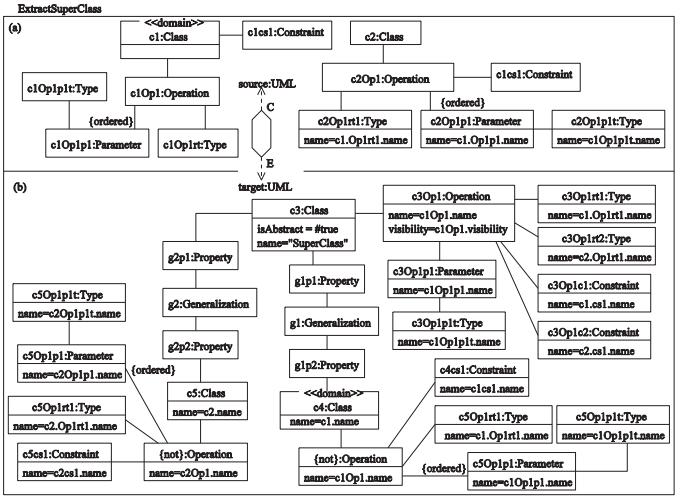


Figure 9: QVT Extract Superclass Transformation

A QVT representation of the extract superclass model transformation is shown in Figure 9. The QVT specification consists of 5 class instances, 5 operation instances, 28 other metamodel class instances, and many links. In contrast, the DBTA specification consists of just a transformation schema with three class schemas, three operation schemas, three attribute schemas, two generalizations; and a source pattern that has two class templates.

The DBTA specification may be more intuitive than the QVT specification for someone familiar with UML. For example, by looking at Figure 7, someone familiar with UML can more easily draw several inferences: (1) a transformation is being performed on class models, (2) a new super class is being created, (3) selected operations are being omitted from classes, (4) the transformation is being performed when two classes have an operation with the same signature. These inferences are easier to draw because DBTA uses the UML syntax and a small set of well-defined directives.

4. Discussion and Conclusions

In this paper we presented a new approach for transforming UML class models. The new approach leverages the UML class model notation and defines transformation schemas that contain imperative statements called directives. The use of the concrete syntax of model elements is an important feature of domain-specific model specification and transformation. The abstract syntax provides for the precise characterization of minute details of a model at the expense of readability and comprehension. The concrete syntax provides for a more abstract representation and the primary expenses is details. Since the concrete syntax of any UML model element specified using the abstract syntax can be specified in DBTA, we believe the use of the concrete syntax provides a viable alternative to the use of abstract syntax in the specification of model transformations. This is akin to developing programs in assembly versus development in a high level language.

The current DBTA specification allows for the addition of other directives should it become necessary. The primary limitation of the new approach is that the approach is class-model specific at this point, and does not support transformations involving other diagram types. We plan on extending DBTA to other UML diagrams (e.g., sequence diagrams and activity diagrams). Part of the strength of the QVT approach is that the level of detail at which transformations are specified allows for the precise specification of many small details. Further research is needed to deter-mine the expressive limitations of using more abstract notations.

As part of our future work, we intend to build a tool to support the DBTA model transformation process. We would also like to explore the use of target patterns, where each target pattern describes the minimum set of properties expected of valid target models. In particular, we are interested in exploring the automated generation of target patterns from a source pattern and a transformation schema.

Acknowledgements

I would like to thank Dr. Sudipto Ghosh and the late Dr. Robert France who guided me through my PhD dissertation and with whom these ideas were first articulated.

References

[1] Jorge Aranda, Daniela Damian, and Arber Borici. Transition to Model-Driven Engineering: What Is Revolutionary, What Remains the Same? *Model Driven Engineering Languages and Systems, proceedings of 15th International Conference, Models 2012*, Innsbruck, Austria, September/October 2012.

[2] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. 2013. A research roadmap towards achieving

scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering* (BigMDE '13), Davide Di Ruscio, Dimitris S. Kolovos, and Nicholas Matragkas (Eds.). ACM, New York, NY, USA, 10 pages. URL: http://0-doi.acm.org.libcat.uncw.edu/10.1145/2487766.2487768

[3] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *Proc. Workshop on Generative Techniques in the Context of Model-Driven Architecture,OOPSLA'03*, Anaheim, California, USA, October 2003.

[4] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[5] J. Greenfield and K. Short. *Models, Frameworks and Tools*. Wiley Publishing, Inc., Chapter 7: Generating Implementations, 2003.

[6] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proc. Model Transformations in Practice Work-shop at Models 2005*, Montego Bay, Jamaica, October 2005.

[7] B. Baudry, T. Dinh-Trong, J. M. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. L. Traon. Challenges for model transformation testing. In *Proceedings of the IMDT Workshop in Conjunction with ECMDA'06*, Bilboa, Spain, July 2006.

[8] D. H. Akehurst, B.Bordbar, M.J.Evans, W.G.J.Howells, and K. McDonald-Maier. SiTra: Simple Transformations in Java . In *Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Model Driven Engineering Languages and Systems, 9th Inter- national Conference, MODELS 2006.*, pages 351–364, Lecture Notes in Computer Science volume 4199, 2006. Springer Berlin / Heidelberg.

[9] J. Zhang, Yuehua, and J. Gray. *Model-Driven Software Development*. Springer Berlin Heidelberg, Book Chapter: Generic and Domain-Specific Model Refactoring Using a Model Transformation Engine, 2005.

[10] W. K. Edwards. *Core Jini (2nd Ed.)*. Java Series. Prentice Hall, USA, 2001.

[11] R. France, D.-K. Kim, S. Ghosh, and E. Song. A UML-Based Pattern Specification Technique. *IEEE Transactions on Software Engineering*, 30(3):193–206, March 2004.

[12] D.-K. Kim. A Meta-Modeling Approach to Specifying Patterns, Ph.D. Dissertation, Department of Computer Science, Colorado State University. 2004.

[13] D.-K. Kim, R. France, and S. Ghosh. A UML-based language for specifying domain-specific patterns. *Journal of Visual Languages and Computing*, 15:265–289, Jan. 2004.

[14] T. Mens and P. V. Gorp. A Taxonomy of Model Transformation. *Electronic Notes In Theoretical Computer Science*, 152:125 – 142, 2006.

[15] M. Fowler. *Refactoring: Improving The design of Existing Code*. Addison-Wesley, USA, 1999.

[16] O. M. G. (OMG). MOF 2.0 Query/Views/Transformations Final Adopted Specification (ptc/05-11-01).

[17] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for composing aspect-oriented design class models. *Transactions on Aspect-Oriented Software Development I*, LNCS(3880):75–105, 2006.

[18] D. Simmonds, A. Solberg, R. Reddy, R. France, and S. Ghosh. An Aspect Oriented Model Driven Framework. In *Proc. Ninth IEEE "The Enterprise Computing Conference" (EDOC 2005)*, volume 0, pages 119–130, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[19] D. M. Simmonds. Transforming UML Class Models, Ph.D. Dissertation, Department of Computer Science, Colorado State University. 2007.

[20] A. Solberg, R. Reddy, D. Simmonds, R. France, S. Ghosh, and J. O. Aagedal. Developing distributed services using an aspect oriented model driven framework. In *Special Issue of the International Journal of Cooperative Information Systems (IJCIS)*, volume 15, pages 535–564, Great Britain, 2007. World Scientific.

[21] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France. Using Aspect Oriented Technologies to Support Separation of Concerns in Model Driven Development. In *Proc. 29th of the Annual International Computer Software and Applications Conference (COMP- SAC 2005)*, volume 1, pages 121 – 126, Los Alamitos, CA, USA, 2005. IEEE Computer Society Press.

[22] The Object Management Group (OMG). Unified Modeling Language: Superstructure. Version 2.0, Final Adopted Specification, OMG, http://www.omg.org, Au-gust 2003.

Adapting Native Applications to Cross Platform Environments

A. Andrew Moubray, B. Tyler Danner, C. Jason Litchfield, D. Harry Sellars, and E. Roger Lee
Computer Science Department, Central Michigan University, Mount Pleasant, Michigan, USA

Abstract *- Computer applications are becoming more and more necessary in everyday life, and their ease of use and portability is becoming increasingly tested with the modern age's wide selection of devices. Countless applications have been created that are used daily, but precious few programs or methodologies have been created that explore the avenue of converting native applications into multi-device, cross platform applications that can be used on a vast array of different systems and architectures. In this paper, we synthesize the work done in this field, and, following the work of S. Amatya, explore avenues of applying our findings to a test case of our own creation. Our work will attempt to demonstrate methods of cross platform translation with the application of current ideas to our single, example program.*

Keywords: Mobile, Hybrid, Cross-Platform, HTML5

1 Introduction

The level of interconnectedness, courtesy of the World Wide Web, the internet, and the recently renamed "Internet of Things" has created an environment in which people consume data at an unprecedented scale. This data, formulated as aggregate information, is valuable to a diverse array of stakeholders, from companies and businesses, to governments and self-promoters. Knowing exactly how, and where, these consumers of information are getting their data is exceedingly important, and more so, how to optimize one's own work to tap into the valuable resource that consumers represent. Unsurprisingly, today's data and the internet is primarily accessed through smartphones, mobile devices, tablets, and other similar machines [1], instead of the dedicated workstations that dominated the world's active user base only years ago. This shift in user behavior has left many corporate, government, and private application developers with a new, and terribly complicated agenda: try to reach as many people - as many consumers - as possible [2]. This paradigm is much easier said than done to adhere to, however, as the level of fragmentation of environments has increased at an impressive rate. No longer are there only a handful of workstation platforms to consider; today, hundreds of variations of mobile devices exist, be they handhelds, phones, tablets, or something else entirely. Developing for each of these devices is challenging for a number of reasons - developers must not only take into consideration the physical device and its dimensions, ergonomics, and general design, but they must also be conscious of the programming languages the devices support, what APIs are available, and even what app store the application will appear in [3].

The ever more fragmented environments has made *de-*fragmentation and optimization of application development very difficult, although it is the goal of this paper to investigate making this a reality. Making the design process of mobile applications more efficient and standardized has become a goal that is becoming more difficult to achieve, yes, although it is also making the achievement of that goal ever more attractive.

To achieve this goal, two areas of fragmentation must be analyzed: the fragmentation of the device, and the fragmentation of the operating system [4]. Looking at the systems and architecture that already exist that support multiple devices as a starting point, an avenue of converting native applications into cross-platform applications becomes clear. Three possibilities offer themselves to this program: developing a bytecode translator between a native and target environment, developing the program in a web-based language, or a hybrid between the two [5].

The overarching goal of this project is to turn our native application into a cross-platform application. By utilizing the research before us, we have determined that with the aid of modern tools and a bit of architectural rework that such a conversion can be accomplished. The tools available include the Unity 5 development engine, Drifty's Ionic Framework for Cordova, and Intel's XDK development suite.

2 Background and Related Work

Cross-platform programming, or historically referred to as heterogeneous coding, or more simply interoperability [4], is a paradigm of program development that, as it should be no surprise, is a relatively recent development. This way of development, that follows the ideology that one should "code once, run anywhere" [6] has evolved from the fragmented nature of today's consumer needs. In his 2013 thesis, S. Amatya analyzed the state of cross-platform development, and his findings expressed a level of improvement that could be applied to the field of Software Engineering.

Cross-platforming has multiple loci of genesis, either from an existing application created in a native environment, or much more common today, a newly envisioned project, ready to be coded with cross-platform deployment from the start. In either case, a number of issues must be first taken into account. The Hewlett-Packard Development Company, in their 2013 whitepaper "Apps to go", outlined key components to take into consideration for developing cross-platform,

mobile applications. They are: ensuring support for multiple operating platforms and form factors of the mobile device(s); ensuring a simplified connection between the application and other systems, including legacy enterprise databases; ensuring 'one-stop-shop' for management and control of the application; and ensuring a consolidated suite for security monitoring [7]. The first two key points are reiterated numerous times by other sources including S. Amatya, each stating that the fragmentation of device, operating system, and data connections must be taken into account [1, 2, 4, 5].

In the first case, native systems, (systems built inside a single language and environment, not originally designed to be ran in multiple environments), can be converted to an application that can run in multiple environments. In every case, commercial products like Titanium by Appcelerator, Rhomobile by Motorola, or PhoneGap can be used to aid in the conversion [2]. Each of these products contain frameworks, ideologies, and of course tools and code to assist in bringing a native code into a cross-platforming environment. Each of these products takes a different approach, which falls into one of three categories: Native, WebApp, or a Hybrid [6].

These three approaches are not unique to native, applications, however, but can be extended to new projects as well, lending themselves to all forms of app development.

2.1 Cross-Platform Approaches

In terms of developing an application for multiple platforms, there are three approaches to consider, in which two of them will provide some level of heterogeneous material. The first is 'native', a term already discussed in this paper. Native development, the standard for much of the computing era, is being slowly abandoned in favor or reaching a wider audience. The benefits of native applications cannot be ignored, however, as they often provide superior speed, graphics, and functionality in general to their Web-Based cousins, making them vastly appealing to enterprises looking for the added level of customization and connectivity to back end, secure databases [1].

The second approach is Web-Based, which leverages the power of the internet and World Wide Web. The Web is a universal standard, accessible by every smartphone and mobile device. The idea behind this approach is to develop an application using HTML5, CSS3, and JavaScript to ensure that every device will be able to run the application. This, although initially appealing, leaves many features and advantages of the native approach behind in a "quantity over quality" way.

The last approach, and the approach most analyzed by both S. Amatya and this research paper, is the Hybrid approach. As the name implies, the Hybrid approach takes an approach of creating a Web-Based app that contains a native app running inside it. This brings the best of both worlds, allowing significantly higher customization and access to backend data, as well as the universal access provided by the Web [6].

2.2 Cross-Platform Toolkits

To make the conversion to cross-platform applications, a number of toolkits and development environments exist that allow for code-once-run-anywhere style development. One such environment is Unity, which is typically known for its game development applications [8]. The benefit of creating applications and systems inside Unity is the ability to push one's completed code to a number of different platforms, including Android and iOS, with only the click of a button. The source code is built, unsurprisingly, on JavaScript or C#, which lends the ubiquity of web-based code to be run on nearly any device.

Another environment that helps developers create heterogeneous applications is Drifty's Ionic Framework [8]. An HTML5 Framework oriented towards the development of mobile apps, Ionic too joins in on the web-based application bandwagon, allowing the developer to pursue multi-platform deployment with only a single source code. As it is based in HTML5, Ionic has the added bonus of being built on a language most developers already know and understand.

The final platform investigated by this research paper is Intel's XDK, or extended development kit. Utilizing Intel's own conversion techniques on top of Cordova, the XDK allows the developer to push their code to a web application that is compatible with most modern browsers, as well as e-readers, mobile devices, and tablets [8].

3 Methodology

3.1 Existing Methods

3.1.1 Summary of the Ionic Framework for Apache Cordova

Ionic is a free and open source library using HTML, CSS, and JavaScript components to build interactive mobile apps, deployable to any web browser and smartphone. Ionic builds on top of the AngularJS framework in order to emulate native application functionality with predefined libraries. The HTML page built with Ionic is run through Apache Cordova to construct an application package that is deployable as a native application [9].

3.1.2 Summary of the Unity 5 Engine

Primarily used as a game engine, Unity provides a robust interface builder, abstracting the layout into objects with their own data attributes and functionality. Unity has diverse deployment options, ranging from a standard web browser, to all modern gaming consoles, e-readers, tablets, and mobile devices. Unity also has native support for SQLite, making the project database imports simple and clean [10].

3.1.3 Summary of Intel XDK

Intel XDK is a development environment created by Intel that utilizes HTML5 for cross-platform development. The IDE allows the development of template-driven, hybrid apps that can be translated and deployed to multiple app stores and form factors. Similar to the Ionic framework, Intel XDK

uses Apache Cordova to construct the native application package to deploy to the specified mobile environment [11].

3.2 Our Approach Using Intel XDK

3.2.1 Overview of Our Cross-Platform Approach

Through the Intel® XDK HTML5 Cross-platform Development Tool, the existing *iPoop* application can easily be ported from the basic, Java-based Android application, to an HTML5 program which can be run on all mobile devices with most web applications. Through the combination of HTML5, CSS, and JavaScript, editing a user-friendly application will be simple and easy to work with. Intel® XDK provides an easy-to-use drag and drop UI builder to assist in this process, allowing the site architecture and paging to be developed before the interface components are refined. Intel® XDK also provides real time layout editing on a web browser or mobile device on the local network. Changes made to the source code are immediately reflected on the synced device or browser, allowing for fast and easy interface changes. The IDE has on the fly application testing, allowing the developer to check for errors prior to the final build, and start running the application as a native installation with an active debugging console. Intel also provides the ability to file share within a work group, and build to app stores directly through the development interface. These features made XDK an excellent choice for the conversion and continued development of the *iPoop* application in a cross platform environment.

3.2.2 Algorithm Design

Fortunately, the inherent nature of computer data structures means that many of the features of the algorithms will remain unchanged from programming platform to programming platform. To this end, many of the basic algorithms of the application will remain unchanged when being moved from a native application to a cross platform application. However, as will be seen below in the implementation, when switching IDE's many issues arise from the use of a native environment versus a cross platform environment, in regards to the manner in which these algorithms are brought to life. A full listing of the necessary algorithms for the basic implementation of the application can be found in the remainder of this section.

Aside from the necessary object classes and their associated set and get methods for the alteration and acquisition of class attributes, the application requires the use of what is known as CRUD methods for database manipulation and access. CRUD stands for "create, read, update, and delete" in terms of database usage. When creating the algorithms for these methods, the developer must take into account the structure of the database in use, in order to create the appropriate algorithm for each method. However, once this process is well known, it is essentially the same process for each method, regardless of the database being used. In the development of the *iPoop* application, a SQLite database is used comprised of three separate tables - bathrooms, ratings, and users. In order to manipulate the data in these tables, the application requires a CRUD algorithm/implementation for each table. This is due to the fact that no matter what language/IDE is used to implement the application all commands must be translated into a fashion that the database will understand.

This database handler, as it is known, will take all data necessities for the application in its native language, and translate it into a language that the database understands. It also will take all data returned from the database and convert it into a form that the programming language for the application will understand. In this way, the database handler operates much like an interim translator, allowing the two objects - the application and the database - to interact despite the fact that they don't speak the same language.

3.2.2.1 Object SET and GET Methods

For each attribute for each class, there exists a *set* and *get* method which all follow the same algorithm regardless of the object or the attribute:

3.2.2.2 Database CRUD Methods

For each table in the database, there exists a set of methods in order to interact with the database in certain ways. The CRUD algorithm, along with basic data structures such as lists, and utilizing computer programming methods such as loops, can be used to create more complex versions of these operations.

Each viewable screen of the application will utilize the given set and get methods from objects, and the given CRUD methods for each table in order to acquire and manipulate the data they need to perform the functions they are designed to accommodate.

3.2.2.3 Login Screen

The login screen will give the user access to the log in, create account, and delete account functions of the application.

3.2.2.4 Map Screen

This screen uses the Google Maps API for operation of the map located on the screen. In addition to this, the screen also contains a method to locate the closest bathroom location.

3.2.2.5 Add Location Screen

This screen allows the user to add new bathroom locations, should they locate one that does not already exist in the application.

3.2.2.6 Rate Bathroom Screen

This screen allows the user to rate their current bathroom location, should they choose to do so.

This covers all of the basic algorithm descriptions for all of the components of the application that require computation in their implementation. In this way, the native application does not differ much from the cross platform application.

However, the divergence will become more apparent once the implementation of the application is examined, beyond the simple algorithm design to run it.

3.2.3 Implementation

The original application was implemented in *Android Studio*, which uses a combination of Java files for control of the program, and XML files for control of the interface/display of the application. For the cross-platform implementation, we required the use of more wide reaching and generally accepted formats than *Android Studio* could supply. To this end, we chose to use Intel® XDK IDE as the development environment for our cross platform application. XDK uses JavaScript and CSS in combination with HTML5 to create interactive applications. This was advantageous as it allowed us the ability to build to any currently used platforms. HTML5 applications are quickly becoming the standard for cross-platform development due to their universality across platforms. What follows is a brief explanation as to the implementation of the cross platform application, in regards to how it was ported from the native environment for the *Android* operating system.

The cross platform application is comprised mainly of HTML5 pages, with corresponding CSS code and integrated JavaScript code for handling of certain aspects of the program. The CSS code is used for formatting of the various aspects of the visible pages, the JavaScript controls the creation, manipulation, and passing of information between components of the application, and the HTML5 pages allow the user to interact with, as well as receive information from the application. We were able to use the same SQLite database that we had used in the Android application, as HTML5 now allows for the use of this type of file. We also utilized a component of HTML5 known as "Local Storage" within the application.

In creating the application in this way, implementation functions in much the same way as creating a web page does. Each page is comprised of an HTML file that controls the interface for that specific page, while a CSS file controls the formatting of the components of the page, and JavaScript controls the programmatic aspects for each function within the page itself. This type of generic usage allowed the application to port more easily between different platforms.

The main menu is comprised of a navigation bar, replacing the navigation drawer of the native Android application. Inside the navigation bar is a list of items, each of which is a link to something else within the application. Clicking the links within the navigation bar redirects to the location specified, in this case to one of the pages that represents a fragment from the old Android application. This page is displayed in a panel located within the main page. By using HTML we are able to construct the pages for each fragment of the application separately, and have a link to them that will display them within the main window.

Each function of the application is contained within its own separate HTML division, which controls the behavior of that function. These pages are called by the main page by clicking on the appropriate link within the navigation bar in the main page. Once that has been done, the page for the associated function is called into a panel on the main screen via its tagged name attribute, where the function can be accessed.

The first page that is loaded upon calling the application is the profile page. This page contains basic information about the logged in user, and is essentially a simple display page for static user data. This page just displays textual and graphical information about the application, and it controls the logout functionality.

The login page grants access to the log in, create account, and delete account functions of the application. This is accomplished through the use of HTML text input boxes and buttons which activate scripts written in JavaScript. HTML utilizes a component known as "Local Storage" which allows applications to store information locally within the application, without the use of external databases. This component was used in this application, as a replacement for the static class that was used to store information in the Android native application. This way, we were able to keep track of what account has been currently logged in to, created, or deleted by the user. Logging in, creating a new account, or deleting an account redirects the user to the appropriate page detailing if their attempts were successful or not, depending on whether they satisfied the necessary criteria for that particular function.

The map screen gives the user access to an instance of a Google Map window with all of the custom locations from the previous Android native application. This is possible due to the fact that HTML5 allows the use of SQLite database in a local instance of the application, in the same way that the native Android application did. The database is accessible within the application through a set of database scripts that control the application's interaction with the database information. This allowed us to populate a map within an HTML5 page in much the same way as we populated the window in the Android application.

The add location screen gives the user access to the add location function of the application. This was created so that users could add new locations to the application should they encounter a location that does not already exist within the application. Again, we utilize text input boxes and buttons in order to give the user the ability to create new locations, which are then added to the existing database, and updated in the map window. This, as always, is only possible if the user meets the criteria to be able to do this, namely, they must be logged in. This is done in an effort to prevent spam, and limit overloading of the database by insert requests from the application.

The rate location screen gives the user access to the rate location function, and also displays the current information for the bathroom location selected most recently from the map screen. The user is able to select a number of stars using the embedded star widget, and then submit their rating by clicking the button associated with it. This sets into motion several

subroutines, including checking to ensure the user is logged in, that a bathroom location has been selected, that the user has not already submitted a rating for that location, and if these are all satisfied, then creating a new rating and adding it to the database. This also requires the new rating to be added to the bathroom's data, and then a recalculation of the bathroom's current rating. Should the user have already created a rating for this location, then the rating would be overwritten in the database, and in the selected bathroom's rating list, which then would be recalculated to determine the current average rating for the bathroom.

The last screen that was created was an empty screen for later use in implementing any custom user options that we want to add to the application. As with the native Android application, we did not implement this portion of the application, as we did not call for it within the requirements for the application, but wanted to leave it available for later development. This window contains no information, but just fills the space until such time as we are ready to implement it.

Use of the Local Storage allows for information to be passed between each of the pages, when windows are changed. In this way we can keep track of what user is logged in, what was the last location selected on the map screen, when new locations are added to the map, and when new ratings have been added by a user, and allow for the system to make corrections to account for these changes.

Thus comprises the general framework of the application, built in much the same fashion as the native Android application was constructed. While the cross platform application was developed using a similar infrastructure of files and connections, the ways in which these connections and files are made varies greatly from the native Android application to the HTML 5 development.

3.2.4 Testing

Testing of the application was performed in a Google Chrome window on a laptop, as well as over mobile devices using the Intel App Preview application on Android and iOS. This was done to show that the application was functional within an environment other than the original native Android environment during development. Given the short amount of time allowed for the process of re-development, not all of the functions of the original application were able to be completed within the deadlines. However, much of the original look and function of the application was able to be implemented and tested.

In keeping with convention on testing a mobile application, we followed the prescribed set of tests for such an implementation. User experience, device compatibility, performance, connectivity, and security testing were performed as expected for such an application.

User experience testing requires that we run the application through its paces, checking for usability and accessibility expectations. We found that the application met all of the usability and accessibility expectations that the original native Android application met.

The device was marginally tested across multiple devices using the App Preview application, however, an actual deployment of the application was not tested as there was not enough time to set up such tests. Porting this application to mobile devices requires extensive knowledge of the device in question, and we did not possess the expertise to do so on such a short time frame. The application was tested on Google Chrome, as well as within the boundaries of the App Preview app on a Galaxy Note 4 and iPhone 6 and performed as expected within the respective platforms.

As for connectivity tests, the application connected and implemented the Google Maps view as expected. There were no issues in connecting to the internet as needed, and connecting to the database as required.

Security testing required that we ensure that all necessary checks and balances within the functions were taken care of, to avoid unnecessary problems with the application. By using a system of checks for logging in, creating and deleting accounts, creating new bathroom locations, and rating bathroom locations, we are able to avoid spamming and attacks of this type by requiring authentication of information to be able to perform these functions. The application passed these tests without issue.

Unfortunately, given the time frame to complete the re-implementation of an entire application, we were not able to finish all of the functionality of the original application. However, the parts that we were able to complete, worked as expected, complied with necessary testing, and fulfilled the same functionality as their native Android counterparts.

4 Results

This section outlines the outcome of the research project and what information can be derived from the work that was implemented.

4.1 Conversion

Converting the application from Android to cross-platform is a very feasible and highly powerful option for developers who wish to create cross-platform applications, or port their native applications over to a cross-platform environment. However, this process still takes about the same amount of time in planning, implementation, and development, so don't expect that converting from one environment to another is going to be a simple, fast, and easy process. Learning the pitfalls of a new set of languages and an IDE can cause the process to take just as long, if not longer, than the original development process for the native application.

We found that, as it often the case, there is no one best way to do something, and that every IDE has its good and bad points, which must be taken into account when deciding what to use to develop a new application. While *Android Studio* showed many difficulties during the implementation process, after using other IDE's we discovered that there were many built in functionalities within the environment that made

implementation of certain functions far easier within this environment than it was within the cross-platform environment using Intel XDK.

However, the simplistic beauty of interface construction using HTML5 far outweighs the downsides of the complexity of data manipulation that were encountered. Android requires a very complex set of classes and functions in order to create a very simple and easy to use user interface. Most of the time spent implementing the original application was spent in refining the interface and making it work appropriately as expected when the proper information was input and the proper buttons were pressed. When creating the application in the cross-platform environment, very little time was spent in creating the graphical interface portion of the application, as every section was build using pre-existing classes within the XDK JavaScript Framework.

This is not to say that implementation in this environment didn't present its own challenges. Due to the use of JavaScript instead of Java, a lot of the robust and deep seeded libraries available to Java are lost as JavaScript doesn't possess the same amount of raw power as a Java application does. There are workarounds, but this meant we were required to come up with more creative ways to implement functions that were very simply implemented within the Android environment.

Overall, we found that converting to cross platform from native development can be relatively simple, and yet complicated. In retrospect, developing cross platform in the first place would have been preferable to the conversion process.

4.2 Device Implementation

By building the application within the XDK Framework and utilizing Cordova, the *iPoop* application is deployed as native installations within the Android, iOS, Windows Phone, or any other mobile/tablet operating system.

The application utilizes the phones native package installer and Cordova allows for utilization of many aspects of a mobile device that are otherwise off-limits to cross-platform development; such as Geolocation features, the accelerometer, and so on. Subsequent builds are capable of being pushed to the app store on each platform and updated to the installed devices.

4.3 Unexpected Challenges

Due to the constraints of the programming language used for the native environment development, when it came time to convert over to the cross platform, many of the modules had to be rebuilt from scratch. This was not something we had expected to be necessary. We had assumed that some minor tweaks would be necessary, but thought that it would not require much in the way of complete overhaul.

Specifically, the module in charge of regulating the conversation between the database and the application required a complete overhaul. In the native Android environment, there was a built in component in the API for the IDE that facilitated many of the necessary methods to construct the database handler. The cross platform IDE, while containing a similar component, did not have as robust a library of methods within said component, so many of the methods had to be built from the ground up, or had to be written in a completely different manner than they were originally constructed.

It was also necessary to rebuild the user interface components from the beginning as well. The way in which the Android environment constructs an interface is far different that the way an interface is constructed in an HTML5/JavaScript environment. To this end, it was necessary that each component of the Android interface, components known as "fragments", which contain the content for each "page" of the interface, be converted to its own HTML5 code subset, which uses a completely different set of construction for its components.

Also unforeseen, was the fact that SQLite databases are not cooperatively functional within an HTML5 environment when dealing with Internet Explorer or Mozilla Firefox as a browsing environment. So, while we had hoped for the application to be completely accessible when developed in a cross platform environment, we were still limited to what browser we could deploy to due to the limitation of using a SQLite database.

5 Conclusion

By converting the application from a native implementation to a cross-platform implementation, we have improved the reach of the application in terms of the number of devices that it is able to function on. We also simplify the software by using common programming languages that do not require specially implemented APIs, making the software easier to code than it was as a native application. In this way, we have found that, despite the complications that can arise from converting the code from one set of languages to another, it is far superior to implement applications using a cross-platform environment, versus a comparable native environment, and worth the effort to convert a native application into a cross-platform application.

6 Suggestions for the Future

For future developments, it would be wise to access more options to ensure the ease of implementing cross platforming. Other options are including but not limited to alternative development environments or databases. Bearing this in mind, considering these alternatives at a far earlier date in development would help efficiency on a large scale, enabling further exploration of deployment opportunities. For example, early development for the *iPoop* application began in *Android Studio* without cross-platform development in consideration. Choosing a cross-platform development-friendly environment would avoid the issue of converting code from one language (if necessary) to another. This limits inevitable bug-fixing.

For the future, we would also like to develop a set of user defined settings, to give the user a more customized

experience. We were not able to do this within the time frame given for the project, but would very much like to add this in the future.

A more complex system for logging in would be preferable, as it would not take much to break the system as it currently stands. A very simple set of authentication functions was used, which most likely could be easily broken.

Other considerations that were discussed during development, but not implemented, were adding sound, and animation to the interface to give it a more dynamic and interactive feel.

References

[1] Charkaoui, S.; Adraoui, Z.; Benlahmar, E.H., "Cross-platform mobile development approaches," *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in* , vol., no., pp.188,191, 20-22 Oct. 2014

[2] Feifei Tao; Yijie Bian; Mingwei Tang, "Research of cross-platform intersystem integration technology based on SOA," *Future Information Technology and Management Engineering (FITME), 2010 International Conference on*, vol.3, no., pp.386, 389, 9-10 Oct. 2010

[3] Bin Zhang; Tian-gang Xu; Wei Wang; Xia Jia, "Research and implementation of cross-platform development of mobile widget," *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, vol., no., pp.146, 150, 27-29 May 2011

[4] Amatya, S. and Kurti, A. 2014. *Cross-Platform Mobile Development: Challenges and Opportunities*. Springer International Publishing.

[5] Yanyan Zhuang; Baldwin, J.; Antunna, L.; Yazir, Y.O.; Ganti, S.; Coady, Y., "Tradeoffs in cross platform solutions for mobile assistive technology," *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on* , vol., no., pp.330,335, 27-29 Aug. 2013

[6] Amatya, S. 2013. *Cross-Platform Mobile Development*. Linnaeus University - Sweden.

[7] HP, 2013. *Apps to go: Six keys to delivering user-driven mobile applications*. Hewlett-Packard Development Company.

[8] Karadimce, A.; Bogatinoska, D.C., "Using hybrid mobile applications for adaptive multimedia content delivery," *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, vol., no., pp.686, 691, 26-30 May 2014

[9] Intel® XDK | Intel® Developer Zone: 2014. *https://software.intel.com/en-us/html5/tools*. Accessed: 2015-04- 13.

[10] Ionic: Advanced HTML5 Hybrid Mobile App Framework: 2015. *http://ionicframework.com/*. Accessed: 2015- 04- 13.

[11] Unity - Game Engine: 2015. *http://unity3d.com/*. Accessed: 2015- 04- 13.

Metrics for Migrating Distributed Applications

Devon M. Simmonds
Department of Computer Science
University of North Carolina, Wilmington
Wilmington, North Carolina, 28403
simmondsd@uncw.edu

Abstract

Managing ever changing technology is a significant challenge that has given rise to the model driven architecture and other efforts aimed at decoupling middleware technology from core business logic in enterprise applications. This paper presents a set of metrics for quantifying the effort required to migrate a distributed application from one middleware to another. The metrics may be used to quantify the benefits of a model driven development approach such as the AOMDE where crosscutting functionality is isolated. A case study that illustrates use of the metrics to migrate a Jini application to CORBA is presented. The results of the case study suggests that migrating an application to a new technology may involve significant human effort and that investing in aspect-oriented development could result in significant benefits.

Keywords: *aspect-oriented software development, distributed applications, application migration, middleware.*

1. Introduction

Software engineering [1] is a central sub-discipline of computer science centered on finding appropriate methods and tools for the systematic development and evolution of complex software systems. Software evolution [2] is itself a challenging undertaking especially when the software being managed has embedded crosscutting technologies [12]. The challenge stems in part, from the pervasiveness of such technology throughout enterprise applications and the resulting difficulty that this tangling and scattering of middleware technology presents when transition to a new middleware is desired. This difficulty in making the transition to new technologies has led to such efforts as the model driven architecture [4] and more recently, to Platform as a Service (PaaS) [8, 9].

The MDA is premised on the availability of principles, techniques and tools for decoupling middleware technology from core business logic in enterprise applications. In MDA, model driven engineering involves separating between platform independent models and platform-specific models and defining functional mappings that capture how PIMs are transformed into PSMs and vice versa. Several successful MDA stories have been reported [10].

Cloud-based platform as a service is a more modern solution to the problem of managing embedded technologies. In PaaS developers need not worry about changing technologies since the services that the technologies provide are now available from a vendor – as a service. This is quite an attractive option, assuming that organizations are free to change their PaaS providers. Against this backdrop, it is easy to forget that there are those organizations still bemused with the changing technology problem resulting from the presence old technology in their legacy applications. For these organizations, the transition to new technologies through an MDA-style approach or through PaaS remains a challenge and therefore, having some mechanism for systematically quantifying such transitions is beneficial.

This paper presents a set of metrics for quantifying the effort required to migrate a distributed application from one middleware to another. A case study that illustrates use of the metrics to migrate a Jini [3] application to CORBA [5] is presented. The rest of the paper is organized as follows. Section 2 introduces the migration metrics and a model for computing migration effort. Section 3 presents a case study of migrating an application from a Jini to a CORBA

platform. Finally, section 4 presents a discussion of the results and their implications.

2. Metrics for Application Migration

When a non-aspect-oriented distributed application is migrated [14, 16] to a new middleware, four activities must be undertaken:

1. The semantics of the new middleware must be learnt. This includes determining when, where and how each middleware feature is used as well as the inter-dependencies between middleware features and the distributed application (server or client). This is an important requirement even in cases where middleware features are pre-packaged as design or code aspects since it may be necessary to correct defects in the development and testing efforts.
2. The old middleware must be removed from the application code. This involves:
   a. Deleting statements from the application in cases where middleware statements do not have embedded business logic code.
   b. Modifying statements in the code in cases where business logic code statements have embedded middleware expressions.

   While this requirement is absent when an aspect oriented approach is used its computation is useful to determine the effort saved by aspect oriented and PaaS approaches.

3. The application code must be refactored [15] to facilitate the integration of the new middleware.

4. Code for the new middleware must be added to the application. This involves:
   a. Adding new statements to the application in cases where middleware statements do not require embedded business logic expressions.
   b. Modifying application statements in cases where business logic statements require embedded middleware expressions.

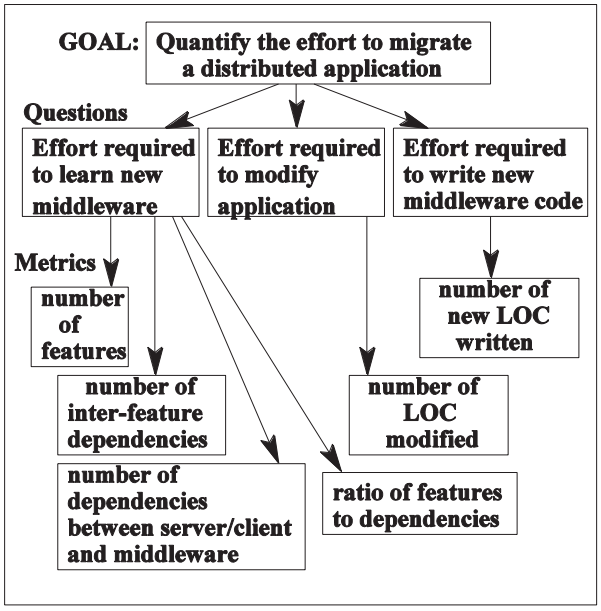


Figure 1: GQM Model for Application Migration

Using the four steps presented in the previous section, a model for quantifying the effort to migrate an application to a new middleware was developed using the Goal-Question Metric paradigm [13] and the concepts and metrics central to the application migration effort were identified. The GQM model is shown in Figure 1 classifies migration effort into two components: *learning* and *development*.

The *learning* element identifies the effort required to learn a new middleware technology. This element is represented by the first question in the GQM model. The *development* element identifies, (1) the effort required to modify the existing application and (2) the effort required to create new middleware artifacts. Each of these items is represented as a question in the GQM model.

2.1. Quantification of Learning Effort

The *learning* element of the model quantifies the effort expended to learn a new middleware technology in order to be able to write and deploy applications using the target middleware. We assume that developers are not familiar with the target middleware. Learning effort (LE) is quantified using the formula:

$$LE = ISD + IFD + NOF$$

where ISD represents inter-service dependencies, IFD represents inter-feature dependencies and NOF represents number of features that collaborate to

provide the middleware services used. Learning effort could apply to both source middleware and target middleware. That is, for legacy systems, the developer may need to learn both the old middleware as well as the new. Middleware features typically involve such things as such as security, transactions, persistence, and distribution. The intuition behind this formula is that while a developer may be familiar with a programming language syntax, the semantics of a new middleware will have to be learnt. In addition, language semantics is a function of structural semantics (interdependencies and when and where a feature is used) as well as algorithmic semantics (how a feature is used).

It should also be recognized that learning the semantics of and writing code for services that have more dependency relationships with other services are expected to be more difficult than services with fewer inter-service relationships. Similarly, a service with more intra-service collaborating features is expected to require more effort to learn and write code. Using this information, and the GQM model in Figure 1, the effort quantification model shown in Figure 2 was developed.

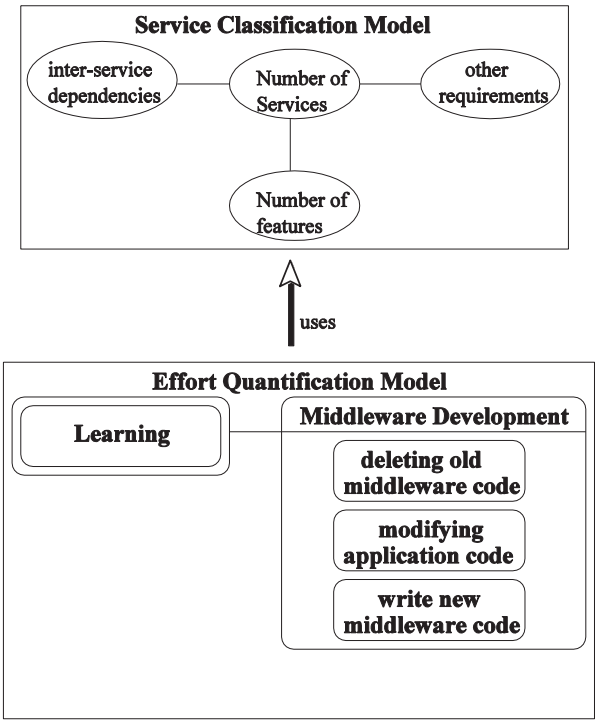


Figure 2. Migration Effort Quantification Model.

2.2. Middleware Development Effort

Middleware development involves middleware coding and middleware modification. The *middleware coding* element is used to quantify the effort required to write code for the new middleware. Middleware coding effort (MCE) is quantified as the number of new lines of code (NLOC) that must be written for the new middleware. The formula used is:

MCE = sum (NLOC added for each method).

The middleware modification element quantifies the effort required to:

1) Delete old middleware code where the middleware code does not have any embedded business logic expressions. For example the Jini statement *ActivationGroup.createGroup (gid, group, 0)* does not quires any embedded business logic expression. The statement is used to create a Jini activation group.
2) Delete old middleware code where the middleware code does not have any embedded business logic expressions. For example the Jini statement *ActivationGroup.createGroup (gid, group, 0)* does not quires any embedded business logic expression. The statement is used to create a Jini activation group.
3) Modify business logic statements in cases where business logic statements require embedded expressions of the new middleware. For example, the explicit propagation of distributed transaction contexts in CORBA make it necessary for methods to have CORBA specific parameters.

The middleware modification effort (MME) is quantified as the number of middleware lines of code that must be changed (i.e., modified or deleted) in each method. The formula used is:

MME = sum(MLOC changed for each method).

The middleware development effort (MDE) is given by:

MDE = MCE + MME

In summary the application migration effort (AME) for a specific application is computed as a combination of learning effort (LE) and development effort (MDE) using the formula:

AME = LE + MDE …….. (1)

where LE = ISD + IFD + NOF …….. (2)

and MDE = MCE + MME ……. (3)

3. Case Study: Jini to CORBA Migration

Using formulas 1, 2 and 3, the effort quantification model requires the computation of the learning and development components.

3.1 Learning Effort Computation

In order to compute learning effort, the number of services used, their inter-relationships and the number of features and their inter-relationships must be computed. Only one service, (`distribution') was used in this case study.

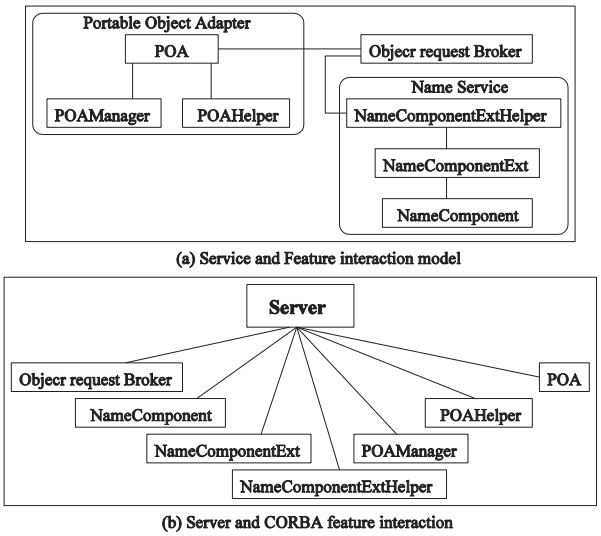


Figure 3. Jini to CORBA Migration Quantification Example.

As a result, inter-service dependency (ISD) is 0. In order to determine learning effort for CORBA distribution, the following metrics must be computed:

1) The number of middleware services used for CORBA distribution. For this case study only one service (`distribution') was used.
2) The number of dependencies among the CORBA features used to effect distribution.
3) The number of dependencies between CORBA distribution features and the business functionality.
4) The number of lines of CORBA code that would normally be written for the migration effort.
5) The number of lines of application code edited or deleted to eliminate Jini code.

The inter-dependencies of the features that collaborate to provide CORBA distribution is illustrated in Figure 3a. The figure shows that a CORBA server requires use of seven different CORBA features. Learning requires an understanding of:

1) The role of the portable object adapter (POA), naming service and object request broker (ORB).
2) The relationship between the POA and the ORB.
3) The relationship between the ORB and the naming service.
4) The relationship between naming service components.

Overall a total of six inter-feature dependencies (IFD) must be understood. In this example the CORBA naming service is used.

The program-feature dependencies (PFD) of the CORBA features and the business functionality is illustrated in Figure 3b. In this case the semantics of seven distinct relationships need to be understood. The learning effort is therefore computed as:

LE = ISD + NOF + IFD + PFD = 0 + 6 + 7 + 7 = 20 units of learning.

This number indicates that before migrating to CORBA, a developer must understand 13 relationships and seven features. This includes comprehending the structural feature semantics (when and where a feature is used) as well as algorithmic semantics (how a feature is used).

3.2 Computing Middleware Development Effort

Middleware development effort involves writing code for CORBA, and modifying or deleting application code with embedded Jini statements. The middleware coding effort (MCE) is quantified as the total number of CORBA lines of code (MLOC) that must be added to an application. An estimate for MCE was computed as the LOC written for the CORBA aspects. This is a reasonable assumption since the functionality provided by the aspects must also be provided by the developed code. The total lines of code written for CORBA server aspects amount to: 26 LOC

for the server and 19 LOC for the client. In summary:

MCE = 26 + 19 = 45 LOC

The total number of lines of Jini code to be eliminated amount to 199. This is the number of LOC written for the Jini aspects. We assume that no middleware modification effort (MME) will be required for CORBA distribution. The total middleware development effort is therefore:

MDE = MCE + MME = 45 + 199 = 244 LOC

In summary the application migration effort (AME) for the application is computed as:

AME = LE + MDE …….. (1)
= 20 units of learning and
244 units of development.

4 Discussion and Conclusion: Making Sense of Metrics

This result suggests that migrating application across middleware may involve significant human effort. This inference is based on the following observations:

1. The learning and development efforts for a typical distributed application will be much more than for this example because a distributed application may use any number of middleware services, for example, transaction, security, events, fault tolerance and concurrency.
2. In this case study we used pre-tested aspects that were pretested and would therefore eliminate some errors that would normally be uncovered during a development project where middleware code is written from scratch.
3. Our model ignores the cost of inter-service connectivity and inter-feature connectivity and their varying degrees of complexity. That is, some inter-connectivity is more complex and requires more time to grasp. For example, understanding distribution is less complex that understanding transaction management.

This simple case study provides some indication of why platform as a service is such a growing phenomenon. However, there are many legacy systems where one or more middleware are endemic and for these organizations making the transition to a new middleware or to PAAS will not occur without some pain.

References

[1] Pressman, Roger. *Software Engineering: A Practitioner's Approach.* New York: McGraw Hill., 2010. Print.

[2] Tom Mens. 2009. The ERCIM working group on software evolution: the past and the future. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops* (IWPSE-Evol '09). ACM, New York, NY, USA, 1-4. DOI=10.1145/1595808.1595809 http://0-doi.acm.org.libcat.uncw.edu/10.1145/1595808.1595809

[3] W. K. Edwards. Core Jini (2nd Ed.). Java Series. Prentice Hall, USA, 2001.

[4] The Object Management Group. The Model Driven Architecture. URL http://omg.org/mda/, 2015.

[5] The Object Management Group. The Common Object Request Broker Architecture CORBA/IIOP URL: http://www.uml.org/.

[6] The Object Management Group. The Unified Modeling Language (UML) URL: http://www.corba.org/.

[7] David E. Bakken. Middleware. URL http://www.eecs.wsu.edu/ bakken/middleware.htm, 2002.

[8] Mitesh Soni. 2014. Cloud computing basics—platform as a service (PaaS). *Linux J.* 2014, 238, pages.

[9] Jin Shao and Qianxiang Wang. 2012. A model-driven monitoring approach for Internetware on platform-as-a-service (PaaS). In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware* (Internetware '12). ACM, New York, NY, USA, , Article 14 , 8 pages. DOI=10.1145/2430475.2430489 http://0-doi.acm.org.libcat.uncw.edu/10.1145/2430475.2430489

[10] The Object Management Group. The Model Driven Architecture Success Stories. URL: http://www.omg.org/mda/products_success.htm,

2015.

[11] Aspect Oriented Software Development. AOSD Webpage. *URL* http://aosd.net/, 2002

[12] Gregory Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Christina Videria Lopes, Jean-Marc Loingier, and John Irwin. Aspect Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer Verlag LNCS 1241*, Finland, June 1997.

[13] V.R. Basili. Software modeling and measurement: The goal question metric paradigm. Technical Report Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), Computer Science Department, University of Maryland, College Park, MD, 1992.

[14] Micro Focus. Application Migration Benefits. *URL* *http://www.microfocus.com/Solutions/Migrate/allbene_ts.asp*, 2004.

[15] Charles Zhang and Hans-Arno Jacobsen. Refactoring Midleware with Aspects. In *IEEE Transactions on Parallel and Distributed Systems*, volume 14, pages 1058.1073, November 2003. 2.3.

[16] Devon Simmonds. "In Support of An Aspect-oriented Approach to Migrating Distributed Applications", *in proceedings of the 1st Caribbean Conference on Information and Communications Technology (CCICT2009), Kingston, Jamaica, March 16-18, 2009.*

SESSION

SOFTWARE ARCHITECTURE AND FRAMEWORKS + ENTERPRISE ARCHITECTURE + COMMERCIAL ISSUES

Chair(s)

TBA

Development of Enterprise Architecture of PPDR Organisations

W. Müller, F. Reinert

Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB

76131 Karlsruhe, Fraunhoferstraße 1

GERMANY

Abstract - *The growing number of events affecting public safety and security (PS&S) on a regional scale with potential to grow up to large scale cross border disasters puts an increased pressure on agencies and organization responsible for PS&S. In order to respond timely and in an adequate manner to such events Public Protection and Disaster Relief (PPDR) organizations need to cooperate, align their procedures and activities, share the needed information and be interoperable.*

The paper at hands provides an approach to tackle the above mentioned aspects by defining an Enterprise Architecture (EA) of the PPDR organization and based on this EA define the respective System Architectures. Based on a methodology which refines architectural artefacts of the OSSAF by using NAF views, a tooling for a lightweight architecture development model is presented.

Keywords: *Enterprise Architecture, Architecture framework, Public Protection & Disaster Relief, NAF, OSSAF*

1 Introduction

Public Protection and Disaster Relief (PPDR) organisations are confronted with a growing number of events affecting public safety and security. Since these events either expand from a local to a regional and to an international scale or are from beginning affecting multiple countries the pressure on PPDR organisations to be able to cooperate in order to respond timely and adequately to such events increases as well. The need of cooperation demands for aligned procedures and interoperable systems which allows timely information sharing and synchronization of activities. This in turn requires that PPDR organizations come with an Enterprise Architecture on which the respective System Architectures are building. The Open Safety & Security Architecture Framework (OSSAF) provides a framework and approach to coordinate the perspectives of different types of stakeholders within a PS&S organisation. It aims at bridging the silos in the chain of commands and on leveraging interoperability between PPDR organisations. In [1] a methodology was presented, which based on the Open Safety & Security Architecture Framework (OSSAF) framework [2] and provided the modeling vocabulary for describing a PPDR Enterprise Architecture.

2 Related work

The goal of Enterprise Architecture design is to describe the decomposition of an enterprise into manageable parts, the definition of those parts, and the orchestration of the interactions between those parts. Although standards like TOGAF and Zachman have developed, however, there is no common agreement which architecture layers, which artifact types and which dependencies constitute the essence of enterprise architecture.

[7] defines seven architectural layers and a model for interfacing enterprise architectures with other corporate architectures and models. They provide use cases of mappings of corporate architectures to their enterprise architecture layers for companies from the financial and mining sector.

A layered model is also proposed by [10]. The authors propose four layers to model the Enterprise Architecture: A Strategy Layer, an Organizational Layer, an Application Layer, and a Software Component Layer. For each of the layers a meta-model is provided. The modeling concepts were developed for sales and distribution processes in retail banking.

MEMO [11] is a model for enterprise modeling that is based on an extendable set of special purpose modeling languages, e.g. for describing corporate strategies, business processes, resources or information. The languages are defined in meta-models which in turn are specified through a common meta-metamodel. The focus of MEMO is on the definition of these languages and the needed meta-models for their definition.

The Four-Domain-Architecture [8] divides the enterprise into four domains and tailors an architecture model for each. The four domains are Process domain, Information / Knowledge domain, Infrastructure domain, Organization domain. Typical elements for each domain are also provided. The authors also provide proposals how to populate the cells of the Zachman framework with architectural elements.

The Handbook on Enterprise Architecture [9] provides methods, tools and examples of how to architect an enterprise through considering all life cycle aspects of Enterprise Entities in the light of the Generalized Enterprise Reference Architecture and Methodology (GERAM) framework.

None of the papers addressing Enterprise Architectures covers the special needs of PPDR organizations with their need on timely cooperation, alignment of procedures, and interoperability needs across different organizations.

3 EA development approach

3.1 Open Safety & Security Architecture Framework (OSSAF)

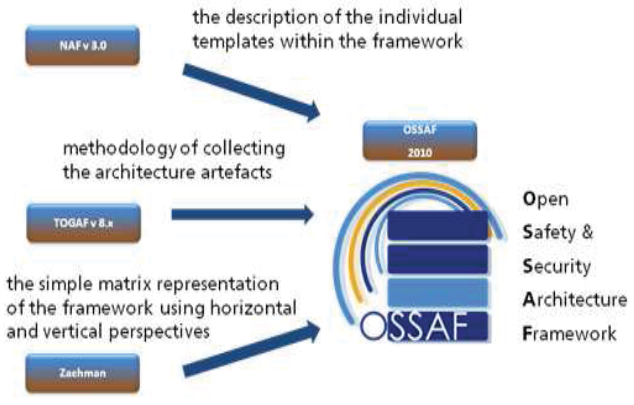
For PPDR organizations, [2] proposes the Open Safety & Security Architecture Framework (OSSAF). The framework incorporates concepts of several mature enterprise architecture frameworks such as the Zachman Architecture Framework (ZAF) [3], the TOGAF framework and the NATO Architecture Framework (NAF) [5] (see Figure 1).

1. The methodology of collecting information and artifacts contributing to the architecture from TOGAF.

2. The two dimensional matrix representation of the framework for structuring the different perspectives from Zachman.

The OSSAF whitepaper [2] also mentions that the NAF meta-model and views may be used where suitable for describing the content of the different perspectives, but does not provide details on the application of the NAF views.

Figure 1: Inputs to OSSAF



OSSAF proposes a total of four perspectives and a total of twenty views. In general it depends on the intention of the architecture under development which views are actually instantiated. In other words the views can be tailored to the specific needs of the architecture under consideration.

3.2 EA development methodology for PPDR organizations

The methodology proposed in [1] for the development of enterprise architecture of PPDR organizations follows a pragmatic approach, looking at an "enterprise" as the joint undertaking of one or more organizations with PS&S responsibilities that operate across a distributed and often complex environment. In this context an enterprise is seen as a nonprofit-oriented organization or complex structures of organizations (inter-organizational aspect of enterprise definition) such as national PPDR organizations, for example national police or fire-fighter organizations.

To handle the task of developing an Enterprise Architecture for PPDR organizations, [1] used the approach of capability based planning. One can understand a Capability according to [1] as:

"An ability that an organization, person, or system possesses. Capabilities are typically expressed in general and high-level terms and typically require a combination of organization, people, processes, and technology to achieve."

Following the capability based planning approach as the overarching guideline; our methodology for the development of an EA proposes scenarios as main input. The first step in the development approach, even preceding the definition and development of scenarios, is the definition of Visions and Goals in order to depict an overall strategy including the winning of supporters for the overall architecting approach.

Since the OSSAF framework already proposes to use NAF views where suitable as templates for describing the OSSAF views and the NAF views defines a vocabulary, [1] used NAF as the modeling vocabulary for describing the OSSAF perspectives and views where suitable.

Table 1 summarizes the general mapping of NAF views to OSSAF perspectives as defined in [1]. Each column represents a perspective defined by the OSSAF framework. The rows represent the views per perspective, each with a specific semantics defined by OSSAF. To the right of each OSSAF perspective the corresponding NAF-views are mentioned which are seen suitable for representing the semantics required by OSSAF. For example to describe the "Capability Planning" view of the "Strategic" perspective it is suggested to use the NAF Capability View-2 ("NCV-2") and Capability View-4 ("NCV-4") view accordingly. In order to describe the OSSAF "Operational Concepts" view of the "Operational" perspective several NAF views form the NAF Capability and Operational descriptions may be used. These are the Capability dependencies View ("NCV-4"), the Capability to organizational deployment mapping View ("NCV-5"), the Operational activity to capability mapping View ("NCV-6") and finally form the NAF Operational

description the High level operational concept description View ("NOV-1").

Another example for the suggested re-use of NAF views in order to describe the required semantics of the OSSAF is given for the "Systems Interface Model" view of the OSSAF Functional perspective. For describing this OSSAF view the NAF Systems descriptions are proposed, especially the System Interface description ("NSV-1"), the Systems communications description ("NSV-2") and the System to System matrix ("NSV-3") view.

The NAF views are modeled with the different elements of the Unified Modeling Language (UML).

The proposed EA methodology is used in the SALUS project [12] to define the Enterprise Architecture of PPDR organizations and the System Architecture of the communication network for those organizations. However, in order to provide an effective usability of the methodology, it was necessary to provide a profile for assigning UML stereotypes and diagrams to the NAF views.

Table 1: Mapping of NAF templates to OSSAF views

OSSAF Views	OSSAF Perspectivec							
	Strategic		Operational		Functional		Technical	
	Vision & Goals	NAV-1 NCV-1	Use Case Scenarios	No proper NAF view	Systems & Services	NSOV-1 NSOV-2 NSOV-3 NSOV-4 NSOV-5 NSV-12	Solution Context	No proper NAF view
	Capability Planning	NCV-2 NCV-4	Operational Concepts	NCV-4 NCV-5 NCV-6 NOV-1	Functional Requirements	NSV-2d NSV-4 NSV-5 NSV-6 NSV-7 NSV-10a	Standards & Protocols	NTV-1
	Funding Model	No proper NAF view	Operational Nodes Model	NOV-2	Systems Connectivity Model	NSV-1 NSV-2a NSV-2b	Device Connectivity Model	NSV-2a NSV-2b NSV-2d
	Laws & Regulations	No proper NAF view	Organization Chart	NOV-4	Systems Interface Model	NSV-1 NSV-2 NSV-3	Product Specification	(NTV-1)
	Local Market Landscape	No proper NAF view	Process Model	NOV-5 NOV-6a NOV-6b NOV-6c			Product Configuration	NTV-3
			Information Exchange Model	NOV-3 NOV-7				

3.3 Tailoring NAF views for PPDR Enterprise Architecture development

The section at hand provides a simplified overview on the core concepts and their relationships as defined in the meta-model of the NATO Architecture Framework (NAF) in order to be used for PPDR EA development.

Figure 2 provides an extract from the overall model used for the development of the PPDR EA. Especially the strategic and operational perspectives of the OSSAF model are depicted. However, for reasons of readability, not all relations, attributes, constraints, and cardinalities are shown.

The model shows that an Enterprise Vision specifies an Enterprise Goal and a Capability contributes to the Enterprise

Vision. A Capability is dependent on another Capability, decomposes into one or more other Capabilities and has one or more Assigned Properties. An operational Node has a Capability and conducts an Operational Activity. An operational Node has a need to exchange information with another operational Node, which is modeled via a Needline which bundles one or more Information Exchanges.

An operational Node is realized by a Resource, either by an Organizational Resource or by a Functional Resource. An Organizational Resource is responsible for an Operational Activity.

The Functional Resource Capability Configuration provides a specific Capability and is delivered via a Configuration Delivery action by a Project Milestone.

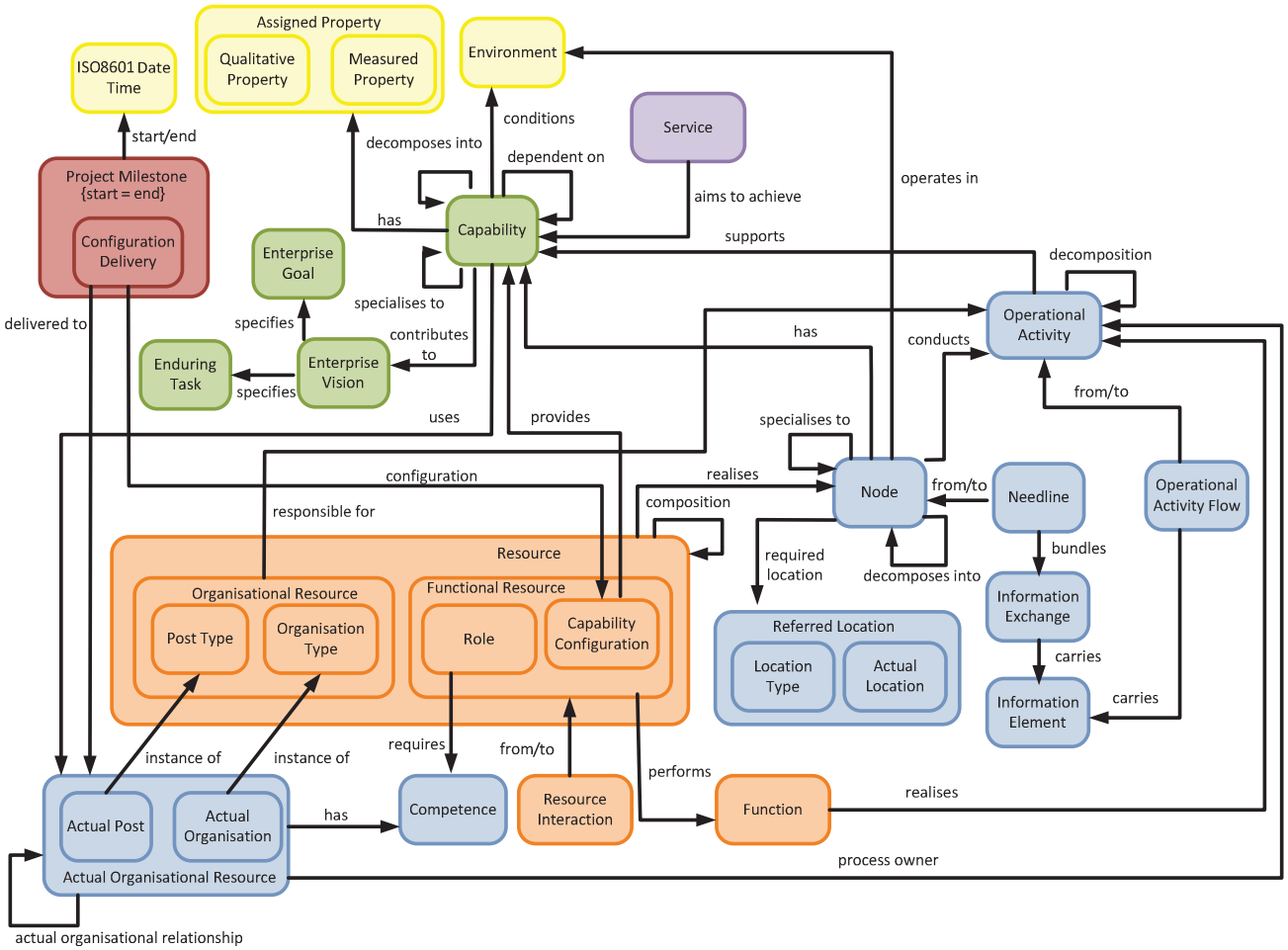
A similar extract from the overall model could be produced also for the Functional and Technical perspectives of OSSAF.

A detailed meta-model description as well as the description of the semantics of each concept and relationship can be found in [5]. It is not replicated here.

The complete profile was produced with the tool Enterprise Architect by SPARX Systems and is based on the MODAF Metamodel 1.2.004 [13].

The MODAF Metamodel was adapted to the needs of EA development for PPDR organizations and extended where needed. An example of such an extension is the multiple inheritance of the model element Node (UML Stereotype Node) from the UML elements “UML class” and “UML part”. This was done in order to re-use the same model element instance across different UML diagrams (e. g. class diagrams and composition diagrams).

Figure 2: NAF Model elements according to the strategic and operational scope



Views - Contents and Representation

According to the approach of describing OSSAF views via suitable NAF-views, the contents of the dedicated NAF-views used in designing the PPDR EA are described. The description contains the model elements captured in the corresponding view (that is actually a section of the overall model), proposes a suitable representation (i.e. graphical, textual etc.) and may give hints in order to support the development of the view under consideration. This is done in a way agnostic to any tool, but refers to UML modeling concepts where suitable.

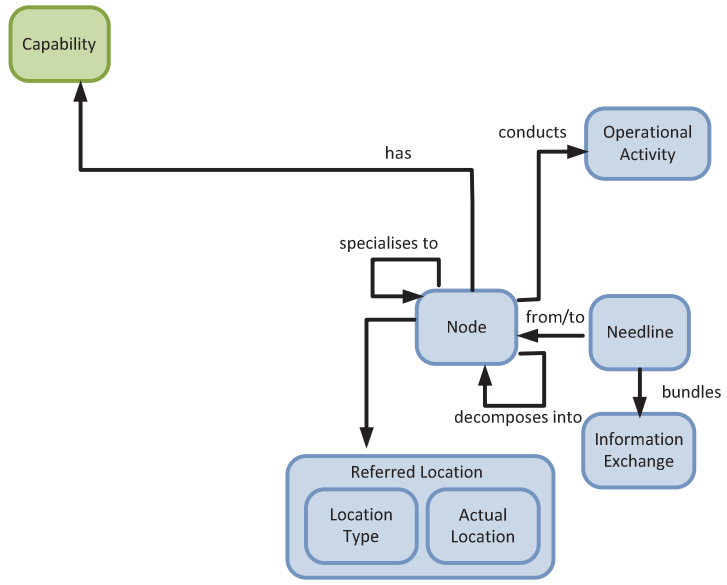
An example of such a description is provided below. It shows how the connectivity between operational nodes and their linkage to capabilities has to be described using the NAF view NOV-2, NATO Operational View, Operational Node Connectivity Description.

NOV-2, Operational Node Connectivity Description

Type of Representation: graphical; diagram which is based on the UML Composite Structure diagram enriched with textual annotations. Needlines describe information flows between nodes (see Hints)

Model elements to be considered: see Figure 3

Figure 3: NAF View NOV-2, Operational Node Connectivity Description



Hint: Node and Needline are recommended, the other elements are optional.

Hint: Depending on the complexity, there may exist several instances of a NOV-2 diagram/table, for example in order to represent nodes with different levels of abstraction (specialisations).

Hint: Exchanges can be annotated (textual) in order to show flows of materiel, energy, or people between nodes as these exchanges are not needlines and therefore do not appear in an NOV-3 view.

Hint: A single Needline represents one-to-many information exchanges (information elements and their attributes).

4 Conclusions and further work

An approach for developing Enterprise Architectures for PPDR organizations was presented. The approach is based on the OSSAF and NAF frameworks. The OSSAF perspectives are described using NAF views. The NAF views are modeled with the different elements of the Unified Modeling Language (UML). In order to provide an effective usability of the methodology, a tool support with a profile for assigning UML stereotypes and diagrams to the NAF views was created.

Based on the Enterprise Architecture, specific System Architectures may be derived.

The proposed EA methodology is used in the SALUS project [12] to define the Enterprise Architecture of PPDR organizations and the System Architecture of the communication network for those organizations.

Acknowledgement: The work described in this paper was partly funded by the European Commission within the European Seventh Framework Programme under Grant Agreement 313296, SALUS - Security And InteroperabiLity in Next Generation PPDR CommUnication InfrastructureS

5 References

[1] W. Müller, F. Reinert "A Methodology for Development of Enterprise Architecture of PPDR Organisations", Proceedings of the 2014 International Conference on Software Engineering Research & Practice (SERP 2014), pp. 259 – 263.

[2] Open Safety & Security Architecture Framework (OSSAF), http://www.openssaf.org/download

[3] Website Zachman Framework, http://zachman.com/

[4] Website TOGAF, http://www.opengroup.org/togaf/

[5] NATO Architecture Framework Version 3, ANNEX 3 TO AC/322(SC/1-WG/1)N(2007)0004

[6] Website Wikipedia, http://en.wikipedia.org/wiki/Requirement

[7] R. Winter, R. Fischer "Essential Layers, Artifacts, and Dependencies of Enterprise Architecture", Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06), IEEE Computer Society, 2006

[8] B. IYER, R. Gottlieb "The Four-Domain-Architecture: An approach to support enterprise architecture design", IBM Systems Journal, Vol 43, No 3, 2004, pp. 587- 597.

[9] P. Bernus, L. Nemes, G. Schmidt (Editors) „Handbook on Enterprise Architecture", Springer, 2003.

[10] Ch. Braun, R. Winter "A Comprehensive Enterprise Architecture Metamodel and Its Implementation Using a Metamodeling Platform", In: Desel, J., Frank, U. (Eds.): Enterprise Modelling and Information Systems Architectures, Proc. of the Workshop in Klagenfurt, GI-Edition Lecture Notes (LNI), Klagenfurt, 24.10.2005, Gesellschaft für Informatik, Bonn, P-75, 2005, pp. 64-79.

[11] U. Frank, "Multi-Perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages", Proceedings of the Hawaii International Conference on System Sciences (HICSS-35), 2002, p. 3021ff.

[12] SALUS: Security and interoperability in next generation PPDR communication infrastructures. http://www.sec-salus.eu/

[13] MODAF Metamodel, https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/63979/20130117_MODAF_M3_version1_2_004.pdf

Enterprise Architecture and Information Technology: Coping with Organizational Transformation Applying the Theory of Structuration

Dominic M. Mezzanotte, Sr. and Josh Dehlinger
Department of Computer and Information Science
Towson University
{dmezzanotte, jdehlinger}@towson.edu

Abstract

Developing large-scale information systems (IS) is never easy or implemented without controversy and impact on an enterprise's stakeholders. Organizational transformation, typically the by-product of new technology and its accompaniment of new processes, frequently manifest itself in ways unforeseen by enterprise management. In many cases, this results in project failure. Historically, enterprises approached IS on a one-domain-one-system at a time solution. This approach has now been supplanted with an enterprise-wide approach to technology with Enterprise Architecture (EA) as the framework used for both requirements and software engineering systems design and implementation. EA embodies a business and technology alignment process aimed at producing an EA plan (EAP) that guides and drives IS development. EA frameworks provide the structured methodologies to support the EAP, yet many EAs fail due to the poor quality in the design requirements used. This paper progresses earlier work analyzing stakeholder behavior and resistance to change proffering a sociologically-driven approach to manage and govern EA design.

Keywords: Enterprise Architecture, Stakeholder Behavior, Resistance to Change, Organizational Transformation

Introduction

In Information Technology (IT), many large organization lean towards a wide range of computer science and computer oriented (techno-centric) frameworks to solve the design and implementation of large-scale information (application) systems (IS) [25][27]. These decades old ontological and epistemological techniques continue to embrace the typical approach to IT from a single domain, one system at-a-time, and on an as needed basis [18]. However, the emergence of a new concept, Enterprise Architecture (EA) and its frameworks (EAF), is slowly replacing the historical IT procedures with a new set of frameworks geared more towards the strategic view and use of information and technology by an organization [18].

As an alternative program to the traditional single IT solution, EA views IT from an enterprise-wide point-of-view moving away from defining and managing a single application domain to a more encompassing and comprehensive strategy that focuses on aligning key corporate IT initiatives with strategic organizational business goals and objective [18][19].

Under the guidance of an Enterprise Information Architect (EIA) that may or may not include participating organization stakeholders, an EA plan (EAP) is produced that contains a detailed description/blueprint of which enterprise functions will be guided through to IT implementation. One of the key components of the EAP is a high-level macro-oriented abstraction of the design artifacts (requirements) defining the various architectures, resources, and infrastructure needed to guide and implement new IT strategies and technology [9][18]. Thus, the EAP contains a synopsis of the organization's "*as is"* operating model and knowledge (explicit and tacit) base and the guidelines needed for developing and providing strategic information for organizational use to support and implement a "*to be*" future business environment [9][12] For many organizations however, EA can be a difficult process that frequently ends in failure.

The organizational context surrounding the transition from an organization's "*as is*" to a "*to be*" state means organizational change brought about by new processes and procedures to be learned by, and the assignment of new roles, duties, and responsibilities to stakeholders. This brings up the typical questions of user (stakeholder) acceptance of EA change and their possible resistance to change. However, a question frequently omitted from this process is the collective impact of EA on the organization.

Simply stated, EA also means change to the organization's character, culture, and structure. Usually hierarchical, each organization arranges its lines of authority, communication, rights, duties, and relationships with internal and external environments which determine the "norms" of how the enterprise does business. Given this context, organizations allocate resources defining which roles, duties, responsibilities, and power are delegated to stakeholders. Thus this activity determines the status of the stakeholder within the enterprise. This, in effect, establishes their power

base, political hierarchy, and societal position within the organization [2][3][20]. As can be seen, these factors affect organizational structure and thus the status quo, equilibrium, and stability organizations strive for and as a result can adversely affect EA design and implementation if not planned for during EA.

In our analysis of literature related to both EA and IT failure, many projects failed for non-technical reasons [5][7][11][24][28]. In fact, the statistics are astronomical in both the number of failed projects and the resources expended in terms of both dollars and time. For example, in the private sector [5][23][28], the failure rate ranges between 66 and 84 percent with the public sector faring even worse [7][11][28] with the failure rate up to 86 percent. Lost dollars are estimated into the hundreds of billions of dollars annually [11][28]. No estimates for time were available.

In this analysis, failure means any project that: is partially implemented, requires extensive rework, exceeded budget and time estimates, and/or is completely abandoned. The factors associated with failure include [5][7][11][23][28]:

- Inadequate executive sponsorship for strategies and associated IT initiatives
- Failure to communicate strategies in a way stakeholders understand
- Lack of stakeholder understanding of what EA represents
- Incomplete/inaccurate stakeholder input of EA design requirements and specifications
- Stakeholder technological incompetence
- Unrealistic time frames and project schedules
- Unclear expectations and objectives for EA.

Clearly, these factors are related to human behavior of one sort or another with blame typically assigned to "poor architecture." In the literature, poor architecture means nebulous, incorrect, and/or ill-defined design requirements [6] [15]. Of interest, the literature cited technology as only accounting for between 4 and 10 percent of the failures [5][28]. In analyzing stakeholder behavior in relation to EA failure, the factors are about evenly distributed between organizational management and the average rank-in-file employee (collectively stakeholder).

Given this premise, our approach to the failure problem considers the possibility that the intersection of organizational transformation, user acceptance of new technology, and resistance to change are inextricably interrelated and intertwined. Our approach to the subject includes accounting for the interaction of the: organization and IT, and how stakeholders will react to EA design and implementation accompanied by organizational transformation.

This paper progresses earlier work by delving deeper into the impact technology has on stakeholder: behavior, acceptance of new technology, and resistance to change. We believe successful EA depends on a stakeholder behavior driven approach that draws attention to stakeholder participation, commitment, and involvement in the EA project. Such an approach encourages and fosters a feeling of stakeholder ownership of EA. In essence, the aim for such an approach is to anticipate, plan for, and identify negative stakeholder behavior and, in effect, implement an avoidance program that allows for corrective action to be initiated early in the process.

In Section 2, we discuss EA, technology, organizational transformation, and their relationship to stakeholder behavior. Section 3 discusses and ties stakeholder behavior and resistance to change using Giddens' *Theory of Structuration* as a lens to guide EA design and implementation. In Section 4, a multi-disciplined stakeholder-driven EAF paradigm is proffered for EA that includes principles and practices from the fields of sociology, psychology, organization theory, and management behavior. In this section, the rules and guidelines are outlined needed to govern, align, and manage IT design through the EA life cycle. Section 5 concludes this paper by discussing the future direction of our work towards a more behavioral-driven solution to EA design and IT implementation.

2. Enterprise Architecture, Organizational Transformation, and Resistance to Change.

Over the past twenty-five plus years, Enterprise Architecture (EA) has emerged as one of the prime frameworks to design and implement complex, multi-dimensional, and large-scale information (application) systems (IS) [9][18]. Given this context, EA represents the genesis for IS and Information Technology (IT) design and implementation. Given this context, *EA defines what IT is to do and IT is doing EA* [15].

In earlier work, deficiencies in existing EAFs are identified from a stakeholder behavioral perspective [15][17]. Though comprehensive and well disciplined from a techno-centric point of view, the current EAF processes pay little or no attention to the impact EA has on organizational and thus stakeholder behavior during any aspect of EA. However, the behavior of both the organization and its stakeholders are inextricably intertwined such that the behavior of each is iterative and recursively reflected in the behavior of the other [8][20].

Given this context, EA can be viewed from both the subjective and objective aspects of human behavior and action and as such affects both the organizational transformation (change) process and the final end-product of EA, the EA plan (EAP) [2] [13]. In the case of EA, it can be argued that the requirements contained in the EAP are best understood as contested and negotiated as iterative interactions between project stakeholders and thus the by-product of their behavior and their feelings

about the new technology. Therefore, the veracity of IT design requirements depends on how stakeholder participate, commit, and get involved during EA to adequately capture, verify, and validate the design artifacts provided EA [24][27]. As can be seen, if these requirements are compromised either intentionally and/or unintentionally, overtly and/or covertly by negative stakeholder behavior, EA and all subsequent IT activity will inevitably fail.

In considering the organization, organizations reflect the culture and character of its stakeholders, from top management through to the lowest stakeholder level (collectively stakeholder). As such, it is more than just the traditional definition of individuals and/or groups of individuals working together to achieve a commonly agreed upon set of business goals and objectives [2][3]. Organizations represent an open system with open boundaries (subsidiaries, divisions, sub-divisions, entities, etc.) and thus a homogenous social-technological (socio-techno) community made up of stakeholders that typically react to internal and external stimuli [2][10].In essence, it is a unique living system/organism that continually strives to evolve, redefining, and maintaining its own recursive identity, character, culture, society, attitudes, beliefs, and hierarchical political and power structure [13][20].

As a stimulus, EA and the technology it introduces into an organization often effect changes to either or both the organization and/or stakeholder. For example, if EA is unexpectedly introduced by management into the organization, the behavioral effect can negatively influence the behavior of all involved stakeholders [3][15][17][20]. However, if adequately planned for and with stakeholder participation and involvement, stakeholder behavior can be positive with stakeholders committed to the process.

While few would argue that stakeholder behavior can be interpreted either subjectively or objectively, the social implications of organizational transformation can result in negative behavioral patterns that often upset the status quo and thus the equilibrium of the enterprise. The predominant logic behind results from forcing stakeholders to accept, adapt to, and take on new roles, duties, and responsibilities without proper preparation can be devastating [4] [20].

As part of an EA planning and organizational transformation process, the following questions should be asked and addressed: What happens when stakeholders are confronted with new and/or enhanced technology (new processes and procedures)? How will stakeholders react to the assignment of a new job, duties, role; and a new set of responsibilities; and to a new societal, power, and political status within the organization?

The first impression one gets from an initial perusal of these questions is the lack of any detailed treatment in providing an answer to any of these questions in existing EAFs. First, organizational transformation is best viewed as both a contradiction and a paradox. For example, let's consider for a moment that EA is a complex phenomenon that requires a pluralistic approach that takes into account not only technology but more importantly the human behavioral components needed for EA design. EA and its EAP then represent the end-product of human action [1][3] [20]. Yet, social theorists posit that the technology introduced by EA can stifle and limit human creativity and innovation [2][3][20]. Thus, the change brought about by EA can be contradictory.

Second, EA can be influenced by human behavior with behavioral patterns that affect:

- The effectiveness of the EAP and the Information System (IS) that depends on the interaction between organizational goals and objectives and the methodologies, principles, and practices (frameworks) used to design the technology
- The implicit and explicit practices, attitudes, beliefs, and values organizations and people typically take for granted, all of which may be overlooked in standard systems (software and requirements) engineering processes.

These actions can be dynamic and lead to four fundamental, interrelated, and intertwined human activities:

- How people create technology
- How people use that technology to accomplish some predefined purpose and/or task
- How technology is introduced into the enterprise
- How people perceive the effect technology will have on their daily lives.

These factors are behavioral and their interpretation by stakeholders form a paradox that has the potential to either positively and/or negatively affect how stakeholders react to the changes brought about by and how they participate and get involved in the design of EA and its EAP.

As can be seen, the usefulness of EA depends on the derived EAP document and the reliability of stakeholder input to that document [14][15]. Given this point of view, the EAP represents a deterministic strategic IT plan based on the sociological and psychological actions and behavior of stakeholders. Therefore, recognizing any activity that can possibly affect the quality of the end-product becomes of paramount concern. Finally, recognizing negative behavioral patterns such as those described above must be considered of prime concern during the EA design life cycle and thus the following questions to be asked:

- How can user acceptance and resistance to change be recognized?
- How can it best be dealt with and handled?
- Can the occurrence be turned to an advantage?

First, resistance to change (resistance and user acceptance) is natural and a part of human nature [6][13][22][30]. It exists in every organization and happens at all levels of the enterprise. Second, if resistance is pervasive or demonstrated by key stakeholders, the result of this behavior will have a detrimental impact on EA. Third and most important, it cannot be ignored or dismissed. Fourth, resistance can come from both the organization and its stakeholders.

From an organizational perspective, resistance can be recognized when management acts by [2][10]:

- Installing structural mechanisms to maintain the status quo and existing equilibrium of the organization
- Implementing procedures and rules that direct people to perform in historical ways
- Maintaining the previous habits, rules, norms, character, culture, and structure of the enterprise.

From the stakeholder point of view, stakeholder resistance includes [6][12][30]:

- Stakeholders acting as creatures of habit continuing to perform and do tasks from existing cognitive behavior without thinking about their actions.
- Coping with change and the complexity associated with new processes and procedures by responding in known ways that lead to resistance.
- Fear that job stability, security, and income will be jeopardized by the new way of performing work.
- Loss of power, influence, and status within the organization.
- Acting in ways that serve their own parochial self interests.

With EA dependent on production of a verifiable EAP, recognizing resistance early in the EA process becomes a critical element in EA design and implementation. However, there is a positive side to resistance to change.

3. Stakeholder and Organizational Behavior: Applying the Theory of Structuration

Enterprise Architecture (EA) and Information Technology (IT) leads to organizational transformation [30]. Given this context, individual stakeholder behavior, including their beliefs, culture, and cognitive life experiences, directly link to that of the central and distinctive characteristics of the organization. These ideas can be expanded and allow resistance to change (resistance) to be treated as a natural and normal human behavioral trait, then organizational transformation can be posited as either a positive and/or negative major shift in the enterprise's operating paradigm, character, culture, and structure. In effect, a change in the way it does business [18] .

The result of this phenomenon can be emotionally devastating to stakeholders as it can alter behavioral patterns that could upset the social, economic, and political hierarchy and structure within the organization. Thus it can be the genesis for failed EA [1] [20]. However, if resistance is treated as human nature and a normal behavioral phenomenon that must be taken into account during EA design, it can be anticipated, planned for, and made a part of any framework(s) desired for EA design that includes aspects of social theory.

Building on the insights of this interpretative approach can lead to increased stakeholder participation, involvement, and thus commitment during the EA life cycle. However, this requires that resistance be planned for and recognized early in and addressed as reality in the EA process. Given this premise, resistance and organizational change can be handled as a means for successful information discourse and exchange.

Given this perspective, Giddens' *Theory of Structuration* allows us to explore stakeholder resistance using several key components of social theory, organization theory, information technology, and communication to analyze resistance from an organization (i.e., structure) and human agency perspective and thus construct a conceptual EA framework. First, knowing what is causing a particular behavior and what is maintaining it must be recognized. Stakeholders are responsible for providing the input and thus the production of output from EA, the EAP [18][19]. Therefore, more attention must be paid to the *people* involved in EA and perhaps less to the processes associated with the technology. Second, only stakeholders can change stakeholder behavior (manager to worker). Stakeholder attitudes, beliefs, and/or cultures cannot be changed as these are integral traits resulting from the cognitive life experiences of the stakeholder [8] [12].

Organizations and humans represent living systems and are each the products of their respective historic life experiences and the environment in which they function. Thus, the way stakeholders interact with the EA process will determines how their commitment to and what they will contribute during EA [3]. Third, what is perceived as resistance is an integral part of the living system that might not be understood [21][22][30]. Thus, helping stakeholders understand the rationale for change and to accept, adapt to, or at least not oppose it, makes stakeholder resistance a critical component that must be reckoned with for successful implementation of IT strategies. Finally, stakeholder beliefs, culture, and values, some of which might be implacable, must be recognized early and dealt with [4][6]. In some cases, stakeholders may be unreasonable. Therefore, mechanisms that either control and/or contain may be

required. From an EA and requirements engineering perspective, our solution addresses resistance with an EA framework tied to and that's formulated, planned for, and designed around stakeholder behavior.

Giddens' *Theory of Structuration* differs from earlier sociological works and dualisms positing a highly complex and abstract property of social systems [8]. We submit at this point that social systems are not structures though they exhibit structural properties that are instantiated as social practices. In this same context, resistance is not a structure but, yet, exhibits structural properties that are implicit, deterministic, and instantiated in daily stakeholder behavior. From this, we can conceptualize resistance as a duality from two dynamically interrelated perspectives: an explicit perspective where resistance is observable and a deeper implicit perspective, recursively linked through the modality of interpretive human actors' actions.

In EA design, the theory can be associated in context with the subjective human experience in the interpretation, creation, and modification of the social world, his *structure* (i.e., organization) [8]. In this case, *structure* is not something concrete but maintains a virtual existence situated in time and space. In addition, though it lacks material characteristics, it cannot exist without human actors who interact in a recursive manner while interpreting its dimensions [8]. In essence, structuration theory represents a social framework where human actors live, work, and interact creating and recreating social culture.

At the same time, and though Giddens doesn't address technology, the theory can be applied to technology depending on how it is perceived and used claiming that social structure can constrain and limit stakeholders' ability to be innovative and creative. Both culture and structure form Giddens' *duality of struc*ture [8]. In this context, stakeholders allow the shared abstractions of social structures to constrain their action and induce behavior influenced by authority relationships and other organizational change. Hence, the absence of material constraints attests to the power of those socially constructed abstractions to elicit behavioral patterns of compliance and conformity. Therefore, Giddens' theory can be applied to EA by recognizing that both structure and agency represent a *duality* and, at the same time, that each is iteratively dependent on each other.

Orlikowski applied IT to the theory formalizing her *Structurational Model of Technology* (SMT) [20] as "in its constituted nature – information technology is the social product of subjective human action within specific structural and cultural contexts – and in its constituted role – information technology is simultaneously an objective set of rules and resources involved in mediating (facilitating and constraining) human action, and thus hence contributing to the creation, re-creation and transformation of these contexts" [20].

4. Mitigating Resistance to Change: Coping with Stakeholder Behavior

Enterprise Architecture (EA) and Information Technology (IT) have come to be synonymous with organizational transformation [2][3][20]. Sometimes this transformation can be easy and in other circumstances, it can be extremely traumatic. However, organizational transformation has become the new norm and certainly a way of life in today's technologically-driven world.

This paper progresses our process of designing and developing a behavior-driven framework for EA that emphasizes a hermeneutic, iterative analysis of social, organizational, and management influences that affect EA design. In this case, exploring and analyzing the effect resistance to change (resistance) has on organizational and stakeholder (collectively stakeholder) behavior during EA and in this ,the hermeneutic analysis. From this point of view, the focus can be directed to the constructive aspects of resistance to alter stakeholder behavior. In this case, altering negative stakeholder behavior means first and foremost recognizing that human stakeholders are emotional beings and creatures of habit [8]. Therefore, recognizing behavioral patterns that exhibit resistance provides a way to change behavior such that accepting change can be healthy.

Given that resistance is constituted behavioral acts that take place from both human and social interaction, these behavioral tendencies and acts can be mitigated by:

- Bringing change to the forefront of the process by explaining the need for change.
- Providing a vision that stakeholders can relate to that helps them to understand why change is necessary for the organization.
- Utilizing those stakeholders as role models who want to "own" and drive the change process. These kinds of stakeholder can help in getting other stakeholders to embrace and accept, or at least not oppose, change.
- Helping stakeholders deal with the emotional aspects of change. Perhaps answering the question "What's in it for me?".
- Creating an environment that appeals to stakeholder self-interest. In effect, waking the talk, not just talking the walk.
- Providing skills training. EA means new processes and procedures that stakeholders must adapt to, accept, and learn. Providing stakeholders with the skills/training they need to work with the new processes and procedures and be successful can negate negative behavior and thus mitigate resistance.

While it might not be possible to control all of the factors affecting EA, we can control the design and implementation processes by instantiating how we will

arrive at a viable IT solution. For example, frameworks can be controlled and allow features to be incorporated that include conducting open-ended interviews, an on-going examination of stakeholder interpretations and expectations, continual analysis of social and work practices to identify risk, and thus avoid approaches that pay little attention to social issues and context.

First and from a structurational viewpoint, identify and involve stakeholders that want to get involved and make them part of the planning and decision-making process. This means asking them for their thoughts about the new technology plans, soliciting their suggestions and then incorporating their ideas in the solution. In effect, listen and act to what's going on.

Second, clearly define the strategic and business-oriented need for change, communicating this information through the organizational levels and soliciting stakeholder thoughts about the plan. Expect there to be defiance – some people will not openly speak about the change but rather ignore it and speak to others about how detrimental the change could be.

Third, plan to deal with the emotional needs of stakeholders by being open and honest about the impact change will have on their place in the process. In effect, handle the *soft people needs* of the stakeholders by involving them and changing only that which needs to be changed. In essence, identify the source(s) and reasons for negative reactions and perhaps modify your assumptions, clarify what is being done, and thus reinforce the rationale for change.

Fourth, resistance should not be confronted from a defensive position. Design flexibility into change by allowing stakeholders to participate in the design process and assimilate new behaviors and redefine their roles during implementation of the EA change process. What we perceive as resistance is really a part of the system we don't understand.

Fifth, do not allow for stakeholders to modify, abuse, or misuse the new technology and thus revert to the previous way of doing work. Focus on the specific positive aspects of change without committing to the process until the organization is ready for implementation.

From this, we can build a sociologically behavioral-driven framework for EA design using Giddens' *Theory of Structuration* that leads to stakeholder participation, involvement, and commitment. For EA to be successful, we need to: 1. know what is causing a negative behavior and what is maintaining it; 2. change behavior recognizing you cannot change attitudes, beliefs, and/or cultures; and, 3. only change behavior that is understood and only if you understand human behavior.

One of the most important outcomes expected from EA organizational transformation [3][18]. The importance of this aspect of EA manifests itself in the translation of EA design requirements into a manifesto containing the planning, alignment, guidelines, and rules for governance leading to a Software Requirements Specification (SRS), document that will be used throughout the IT design and implementation life cycle [24]. [31] Yet, the EA process highlights one of the most difficult and arduous tasks confronting the Enterprise Information Architect (EIA), ensuring organizational transformation takes place in an orderly and controlled environment focused on organizational expectations [7][11][22].

5. Discussion, Future Direction and Closing Remarks

The initial stages of Enterprise Architecture (EA) design today are more concerned with the how the technical aspects of Information Technology (IT) design activity is conducted rather than with preparing the organization for EA. This approach differs from traditional approaches in that EA is examined from an organizational and stakeholder behavioral perspective with an EA framework that includes and takes into account these behaviors.

In earlier work [16], we proposed a framework that includes several principles and practices from the fields of sociology, psychology, organization theory, and management behavior. This framework is incomplete. Future work planned continues more research into the behavioral aspects of EA design and to enhance our framework with features that include:

- Establishing and understanding organizational boundaries, domains.
- Choosing a preliminary IT design concept.
- Organization and delegation of EA design activities based on stakeholder skills.
- Coordination of the IT design activities into a single strategic EA plan (EAP).
- Integration and consolidation of IS design efforts into a single comprehensive overall.

6. References

[1] D. Leonard-Barton and I. Deschamps,*Managerial Influence in the Implementation of New Twechnology*, Management Science, Oct, 1988, 34, 10: ABI/INFORM Global.

[2] M. Beer. *Organizational Behavior and Development*, Harvard Business Review, Harvard University, No Date.

[3] M-C Boudreau and D. Robet, *Enacting Integrated Information Technology:A Human Agency Perspective,* Organization Science, Vol.16, No. 1, pp 3-18, January-February, 2005

[4] C. Brooke (various), *Critical Management Perspectives on Information Systems*, Butterworth-Heinermann, Elsevier, Lincoln House, Jordan Hill, Oxford OX2 8DP, UK, 2009.

[5] A.M. Croteau, F. Bergeron, *An Information Trilogy: Business Strategy, Technological Deployment, and Organizational Performance*, Journal of Strategic Information Systems 10 (1001) pp 77-99, April, 2001.

[6] F. D. Davis, *User Acceptance of Information Technology: System Characteristics, User Perceptions, and Behavioral Impacts*, Int. J. Man-Machine Studies, Acdemic Press Limited, 1993.

[7] R. Gauld. "Public Sector Information System Failures: Lessons from a New Zealand Hospital Organization," *Government Information Quarterly*, 24(1):102-114, 2007.

[8] A. Giddens. *The Constitution of Society: Outline of the Theory of Structuration*, University of California Press, 1984.

[9] J. C. Henderson and N. Vankatraman, *Strategic Alignment: Leveraging Information Technology for Transforming Organizations*, IBM Systems Journal, Vol. 32, No. 1, 1993/1999.

[10] T. Herrnes, *Studying Composite Boundaries: A Framework of Analysis*, Human Relations, Vol. 57(I):9-29, The Tacistock Institute, Sage Publications, London, Thousand Oaks, CA, 2004.

[11] B. Lawhorn. *More Software Project Failures*. CAI, March 31, 2010.

[12] R. Lewin and B. Regine. "Enterprise Architecture, People, Process, Business, Technology." Institute for Enterprise Architecture Developments [Online], Available:http://www.enterprise-architecture.info/Images/ExtendedEnterprise/ExtendedEnterpriseArchitecture3.html.

[13] M. L. Markus, *Power, Politics, and MIS Implementation*, Communications of the ACM, Vol. 26, No. 6, June, 1983.

[14] D. M. Mezzanotte, Sr., J. Dehlinger, and S. Chakraborty, *Applying the Theory of Structuration to Enterprise Design*, IEEE/WorldComp 2011, SERP 2011, July, 2011.

[15] D. M. Mezzanotte, Sr. and J. Dehlinger, "*Building Information Technology Based on a Human Behavior Oriented Approach to Enterprise Architecture*," 2013 World Conference in Computer Science, Computer Engineering and Applied Computing, IEEE/WorldComp 2013, SERP 2013. July, 2013.

[16] D. M. Mezzanotte, Sr., and J. Dehlinger, "Enterprise Architecture: A Framework Based on Human Behavior Using the Theory of Structuration." *International Association of Computer and Information Science, 2012 IEEE/ACIS 10th International Conference on Software Engineering Research, Management, and Applications*, 2012.

[17] D. M. Mezzanotte, Sr., J. Dehlinger, and S. Chakraborty, *On Applying the Theory of Structuration to Enterprise Architecture Design*, IEEE/ACIS, August, 2011.

[18] D. Minoli. *Enterprise Architecture A to Z*, CRC Press, New York, 2008.

[19] The Open Group, *TOGAF Version 9*, The Open Group, 2009.

[20] W. Orlikowski. "The Duality of Technology: Rethinking the Concept of Technology in Organization," *Organization Science*, 3(3):398-427, 1992

[21] G. Riva, *The Sociocognitive Psychology of Computer-Mediated Communication: The Present and Future of Technology-Based Interactions*, Cyber Psychology and Behavior, Vol. 5, No. 6, Mary Ann Liebert, Inc., 2002..

[22] S. Rivard, B. A. Aubert, et al, *Information Technology and Organizational Transformation: Solving the Management Puzzle*, Elsevier Butterworth-Heinemann, Linacre House, Jordan Hill, Oxford, OX2 8DP, 800 Wheeler Road, Burlington, MA, 01803, 2004.

[23] S. Roeleven, Sven and J. Broer. "Why Two Thirds of Enterprise Architecture Projects Fail," *ARIS Expert Paper* [Online], Available: http://www.ids-scheer.com/set/ 6473/EA_-_Roeleven_Broer_-_Enterprise_Architecture _Projects_Fail_-_AEP_en.pdf.

[24] N. Rozanski and E. Woods, *Software Systems Architecture*, Addison-Wesley Professional, 2006.

[25] W. Scacchi, *Process Models in Software Engineering*, Final Version in *Encylopedia of Softwre Engineering*, 2nd Ed., John Wiley & Sons, Inc., New York, NY, December, 2001.

[26] S. Spewak, Enterprise Architecture Planning: Developing a Blueprint for Data, Applications, and Technology, J. Wiley & Sons, Inc., New York, NY, 1992.

[27] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide,* John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex, PO19 1UD, England, June, 2000.

[28] The Standish Group, *The Standish Group 2014 CHAOS Report*, 2014 Project Smart, 2014.

[29] B. van der Raadt, S. Schouten, and H. van Vliet, *Stakeholder Perception of Enterprise Architecture*, ECSA, 2008, LNCS 4292, Sprinter-Verlag, Berlin, Heidelberg, 2008.

[30] J. G. Wojtecki, Jr. and R. G. Peters, *Organizational Change: Information Technology Meets the Carbon-Based Employee Unit*, The 2000 Annual: Volume 2, Consulting, Jossey-Bass Pfeiffer, San Francisco, CA, 2000.

[31] J. A. Zachman, *Enterprise Architecture Artifacts vs. Application Artifacts*, Z|FA, Zachman Institute for Framework Advancement, www.zifa.com, No date.

Proposed Framework for Handling Architectural NFR's within Scrum Methodology

Ahmed E. Sabry[1], Sherif S. El-Rabbat[2]

[1]*Computer and Information Systems Department, Sadat Academy, Cairo, Egypt*

[2]*Research and Development Department, BG, Kuwait*

Abstract— *Most of practiced agile methodologies focus on delivering the functional requirements with higher priority and delivery push, lead up to a lot of instability and architectural severe non-compliances and variations. This will be very costly to change in the future phases or sprints and it will be too late to consider addressing main NFR's required.*

This research paper aims to provide architectural refactoring framework and techniques for achieving required levels of NFR's through formalizing Spikes and DoD's within Scrum practices. In addition, it aims to develop a framework for handling different types of technical debts within Scrum managed software products and projects development. Scope covers addressing NFR's within software development through DoD's and techniques for identifying them. Furthermore, finding the best tactics for implementing the NFR's within the Scrum methodology through spikes and refactoring with usage of constraints and rules mainly within DoD's [1, p. 1]. *It has been concluded that incorporation of non-functional requirements should be explicitly defined by including them in the Definition of Done. In addition, adopting the engineering practices such as TDD, CI, refactoring for maximizing the gained value from Scrum.*

A data exported from Jira agile management tool covering nine software products with more than 7000 defined user stories developed using Scrum methodology has been analyzed using data mining techniques.

Keywords: *Non-Functional Requirements (NFRs), Definition of Done (DoD), Software Architecture, Quality Attributes (QAs), Scrum Methodology, Data Mining (DM), Spikes, Refactoring, Continues Integration (CI), Test Driven Development (TDD)*

1 INTRODUCTION

True agility can be achieved through the ability to apply changes into the product in an easy, fast, and flexible way. Within that sense, the organizational agility is constrained by technical agility [2]. That is best reached by knowing more about systems unknowns. As known "unknowns" are better than unknown "unknowns", the architecture play main role over revealing those unknowns, unplanned, and usually implicitly assumed non-functional requirements. Practiced agile methodologies and specially Scrum mandate quite push towards delivering the functional requirements without real considerations for the NFR's. Consequently, architectural constraints and design rules could not be well addressed at least in early stages. The higher priority given to functional specifications leads up to a lot of instability and architectural severe loss of alignment, non-conformances and variations. This is usually very costly to change and it will be too late to be considered in later phases [3]. Because of the importance of NFR's, it is critical that they be considered during early architecture design. It has been addressed that architects commonly consider them simultaneously [4, p. 1737] with focus in later stages. However, architects risk making architectural decisions concerning which tactics to implement and it could be difficult to implement correctly and control.

1.1 Problem Definition and Objectives

Complexity to address the NFR's implementation within the Scrum practices engineered software. This increase the cost of refactoring for achieving those requirements later. It aims at the following:

- Architectural Refactoring for achieving levels of NFR's through formalizing Spikes and DoD's
- To develop a framework for handling technical debts within agile concepts
- Scope will cover addressing NFR's within software development through DoD.
- Identify the best NFR's implementation techniques within the agile concepts (spike and/or refactoring) through (DoD, Constraints, Rules)

1.2 Originality and Value

This research studies how to provide better architectural refactoring techniques for achieving required levels of NFR's. In addition, proving and improving results through applied case-study data.

1.3 Structure of the paper

It starts by reviewing similar researches that researches addressing non-functional requirements in the definition of done within scrum and generally agile practices. Then it

discusses the framework with case analyses models, data collection along with the results. In the following sections, results discussed, relevant conclusions drawn as well as future work.

2 BACKGROUND

While non-functional requirements or quality attributes, as they are also called, may not be sexy, they are still important: They impact the user experience, and they influence architecture and design decisions [5, p. 1]

As important as they are, non-functional properties are sometimes overlooked, particularly in an agile context where we spend less time with upfront research and analysis. This can be particularly painful for those attributes that apply to the entire product [5, p. 1]

Scrum does not help with non-functional requirements, but that does not mean that we are supposed to ignore them [6].

Within Scrum projects, the focus mostly given to implementation for the next set of features from the prioritized backlog items. While features are not everything, work on fixing bugs, usability, performance, and scalability issues shall be planned within sprint backlog including other NFR's [6].

3 NFR AND SOFTWARE ARCHITECTURE

As Non-Functional Requirements (NFR) specifies "how well" the "what must behave" [1], the development team is a great partner for finding relevant nonfunctional requirements, particularly if the team members have worked on a similar product before or deal with support and production issues. Otherwise, inviting members of the operations and customer services group should be considered to listen to their views on qualities such as robustness and availability [5, p. 1].

3.1 TYPES OF TECHNICAL STORIES

Sometimes it is useful to identify different types of technical stories. Mostly because it gets your team thinking at different levels about all of the needs, they might have to be properly implement within the application [5].

Table 1: Types of technical stories

Product Infrastructure	Stories that directly support requested functional stories. This could include new and/or modified infrastructure. It might also identify refactoring opportunities, but driven from the functional need.
Team Infrastructure	Stories that support the team and their ability to deliver software. This may include using of tools, testing, metrics, design, and planning.
Refactoring	Stories that identify areas that are refactoring candidates. Not only code needs refactoring, but also it can often include designs, automation, tooling, and any process documentation.
Bug Fixing	Either clusters or packages of bugs that increase repair time or reduce aggregate of testing time. So this is more of an efficiency play.
Spikes	Research stories that will result in learning, architecture & design, prototypes, and ultimately a set of stories and execution strategy that will meet the functional goal of the spike. Spikes need to err on the side of prototype code over documentation as well, although I do not think you have to "demo" every spike.

3.2 NFR Constraints

As constraint is a condition to make the requirements in line with quality expectations, NFR can be addressed within defined constraints. This helps determine whether the shippable product have satisfied the non-functional requirements or not. Any backlog item may be constrained by NFR. Usually a constraint implemented either during the implementation by the developers (internal quality) or at run-time by the software (external quality) [1]. Therefore, NFR's constraints could be classified into two main categories:

The first category is internal qualities that mainly cover the design-time qualities. This category is addressed through rules. The rule can be defined as "constraint" that sets a limit to comply during software construction. Internal quality attributes may include maintainability and testability, that are barely visible by the stakeholders but simplify how to build the software. The user story could not be considered done until each rule is confirmed by doing peer reviews or inspection. This category may also include Simplicity, Maintainability, Testability, Portability, and Extensibility [1] [5].

The second category is external qualities that mainly cover runtime quality attributes. This category addressed through restrictions. Restriction it can be defined as is a "constraint" that sets a limit to comply during software execution. External qualities may include performance, correctness, security and usability. These may perform the software functions at run time; hence, they are desired while invisible to most of stakeholders. It can be measured for compliance by conducting suitable testing mechanism and automated testing techniques. Restriction is specific for a scenario, and it should have measurable quality objective. This category may include Correctness, Performance, Reliability, Robustness, Scalability, Security, and Usability [1, p. 2].

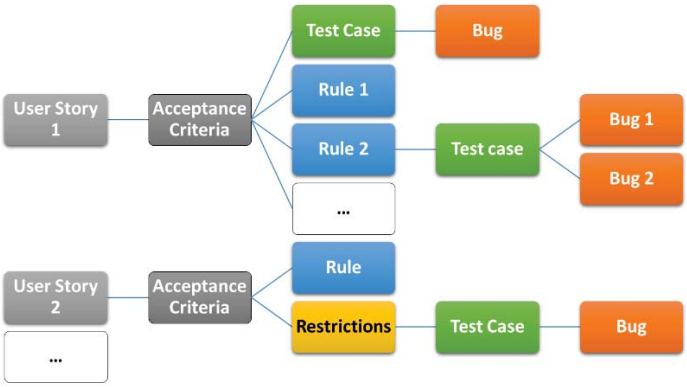


Figure 1: NFR Constraints Illustration

Figure 1 illustrates the relationships among stories, acceptance criteria, rules, restrictions, test cases, and bugs [1, p. 4]. NFR requirements identification and analysis should start with identifying the stakeholders of the product by arranging a meeting, in which the goals of the system should be identified and discussed. The stakeholders shall be a product owner, system administrator, user, architect of the system, and quality manager. With stakeholders, the team will identify the goals, sub goals of the system with the nonfunctional requirements. These sub goals Nonfunctional Requirements will be identified during the meeting [1, p. 6].

It is better in to create a separate user stories for the NFRs in the product backlog, instead of using the user story itself with NFR parameters.

That helps in the prioritization according to the FR user stories and the NFRs user stories sizing because most NFRs can be split throughout different sprints. The background of this type of estimation is along with the functional story development, NFR parameters should also be considered [1, p. 11]. It is critical to address the required NFR's for entire product or important features in the early stages of engineering, this supports creating a greater user experience and make more rational architecture and technology decisions. Capture the requirements precisely to ensure testability [1, p. 15]. Most of modern service systems' development approaches focus mainly on the system facing quality requirements (late NFRs), while more focus should be given to the customer facing quality requirements (early NFRs). NFRs handling remains a challenge, despite of the latest advances in the state of the art and practice of software development [7, p. 7].

There are different views for the NFR's importance, on addresses the types of requirements that affect overall fitness for use given acronym FURPS, which is a requirements categorization for Functionality, Usability, Reliability, Performance, and Supportability. The following activities must explicitly consider NFRs [8]:

- The business relevant feature or epic
- Developing and analyzing prospective architectural features and architectural epics
- Reflect the increased solution domain knowledge using model and architectural refactoring.

Properly defining NFRs requires consideration of the following criteria listed in Table 2 [8, p. 3].

Table 2: NFR's main criteria considerations

Bounded	Some NFRs are irrelevant (or even impairing) when they lack bounded context.
Independent	NFRs should be independent of each other, so that they can be evaluated and tested without consideration or impact of other system qualities.
Negotiable	Understanding the NFRS business drivers and bounded context mandates negotiability.
Testable	NFRs must be stated with objective, measurable and testable criteria, because, if you cannot test it, you cannot ship it.

Sometimes the NFR must be implemented all at once, other times the teams can take a more incremental approach (story-by-story path) [8, p. 3].

Collaboration of the System Team and Agile Teams to create a more practical NFR testing strategy (Diagram) [8, p. 3].

A research [9] drawn a practical decision model for software architectural implementation tactics based on a given specific set of quality attributes requirements. It lists an evaluated set of NFR's as listed in Table 3. It validated the results through triangulation of methods including conducted survey.

Table 3: Description for Main Evaluated NFR's

NFR	Description
Reliability	Capability of the software product to maintain a specified level of performance used under specified conditions.
Usability	How easy it is for the user to accomplish a desired task and the kind of user support the system provides.
Maintainability	Capability of the software product being modified
Testability	Capability of the software product to enable validation over changes
Portability	Capability of the software product to be executed over one environment to another.
Reusability	Ability of easily reused, it depends on degree of dependency among components and it is better to be less
Modifiability	How ease with which it can be modified to changes in the environment, NFR requirements or functional specifications
Performance	Degree to which a system or component accomplishes its designated functions within given constraints, such as time, accuracy, or memory usage.
Security	System's ability to resist unauthorized usage while still providing its services to legitimate users
Availability	The limit of the probability that the system is functioning correctly at time (t)

As part of its results [9] a concept of "safe-tactics recommended to be used for implementation of required set of NFR's with specified fuzzy levels Table 4.

Table 4: concept of "safe-tactics recommended to be used for implementation of required set of NFR's

Tactic Name	Rank
Component Replacement	1
Anticipated Changes	2
Generalize Module	3
Reconfiguration	4
Time Strap (stamp)	5
Record /Playback	6
Restoration	7
Fix the Error	8
Timeout	9
Limit Exposure	10
Removal from service	11
Authorize Users	12
Passive Redundancy	13
Heartbeat	14
Authenticate Users	15
Redundancy	16
Shadow	17
Voting	18
Specialized Access Routines/Interfaces	19
Ping/ Echo	20

Efficiently testing nonfunctional requirements requires some thought and creativity, as otherwise high-cost heavyweight tests may increase the risk of substantive technical debt, or worse, system failure [5].

4 SCRUM METHODOLOGY AND NFR'S

The increasing systems and software complexity with addition to increasing competition in the software industry mandates the need to consider NFRs as an integral part of software development process [10, p. 13].

Within Scrum process, there is a serious risk that non-functional requirements will be neglected, resulting in poor-quality software. It is natural for the Product Owner to think about progress in terms of user stories and features, and to hold the development team responsible for the 'quality' of the resulting software [6, p. 1].

It is recommended to inject non-functional requirements into a Scrum sprints by emphasizing them in the Definition of Done (DoD) plus any associated reviews. Consequently, non-functional requirements for things like reliability and usability can be addressed into user stories [6, p. 1]

Once the sprint starts the development of functional stories with the NFR parameters begin developing and completed with the functionally tested, including the NFR at the end of the sprint [1, p. 13].

During the sprint or in the beginning of the sprint, if any constraints have been identified (in other words if any NFR implementation affects one or more user stories) then it should be added into the product backlog. With addition of a constraint, the system stories in the product backlog need to comply with this constraint may also need resizing if there is an effort required to comply with the constraint [1, p. 13].

In other words, all estimates for future stories will need to take into account the fact that the constraint must be complied with in order to call the story "done".

For Scrum teams, it's important to adopt Agile engineering or development practices that will enable the team to have stable velocity while meeting the quality standards. Engineering practices need to use common tools and frameworks for greater efficiency and tracking. Typically, the engineering practices are captured in DoDs [11, p. 7].

4.1 Unit Testing and Code Reviews

A unit test is a piece of code written and maintained by the developers that exercises other areas of the code and checks the behavior. The result of a unit test is primarily binary, either pass or fail. Developers write a large number of unit tests based on the functions and methods in the code. Unit tests are usually automated and can be run frequently as the code changes. Unit testing frameworks are used to write, maintain and execute the unit tests in an application. Unit tests are written as part of development or coding and this activity is typically part of Story DoD. NUnit and JUnit are two popular tools for unit testing. Some Agile teams refer to unit testing as developer testing. Code reviews and pair programming are popular in agile development teams; many agile experts believe that effective use of code reviews and pair programming can improve the overall quality of the application by identifying and eliminating defects during the coding or development phase itself. The book "How Google Tests Software" emphasizes the importance of code reviews in the development process. It states that Google centers its development process on code reviews. There is far more fanfare around reviewing code than there is about writing it [11, p. 7].

4.2 Test Automation

Building a robust test automation framework is critical for agile teams to deal with regression issues. Test automation tools are used to automate functional, integration and system tests. Automated regression tests can find issues faster and help the team move faster by saving time spent in manual testing. Automated tests can be created as soon as the Story is ready for testing. The team can keep this activity as part of the Story DoD [11, p. 7].

4.3 Continuous Integration (CI)

Test Automation in and of itself does not offer much value to agile teams if the tests are not run frequently to find defects. A Continuous Integration model allows teams to check in code and relevant automated tests frequently, sometimes multiple times a day. Once the new code is checked in and a working build is created, the automated regression tests can run on the build and find the defects faster. As soon as the Story and a relevant test automation scripts are developed, they can be checked into the CI model. Again, this activity can be tracked in the Story DoD. The Story is not done until the automated tests are created and added to the CI and

regression tests are run without any failures. Automated Deployments, Integration Testing, etc. [11, p. 8]

4.4 Test-Driven Development (TDD)

Test-driven development (TDD) is a software development process in which the developer writes an automated test case that states the desired behavior; the test should be initially failing. Then the developer writes the minimum amount of code to pass that test. The new code should be refactored to acceptable coding standards. TDD, which is intended to build the code right, as an engineering practice is encouraged in some Agile circles for better design and improved quality [11, p. 8].

4.5 Acceptance Test Driven Development (ATDD)

ATDD (build the right code) is a complete paradigm shift from other agile software development practices. In ATDD, developers build the application code based on User Stories and acceptance tests and automated tests are run on them to capture feedback from Users and product owners as the development is still in process. The automated tests are defined by product owners and Users using a WiKi mechanism, and then an ATDD framework like FitNesse or Cucumber is used to integrate the tests to working code using fixture code. ATDD integrates developers, testers and product owners and Users into the development effort in a kind of forced collaboration. ATDD encourages team's acceptance of quality as everyone's responsibility [11, p. 8].

4.6 Performance and Load Testing

Performance and load testing are important aspects of testing that ensure the application meets the performance expectations of the Users under typical production load. It's not always efficient to run performance tests for every Story, but they should be run for key features to ensure that response times are below the expected thresholds as defined in the Acceptance Criteria. Performance tests can be run at the end of each Sprint when a set of stories are completed by the development team. Performance testing activity can be part of Sprint Definition of Done [11, p. 9].

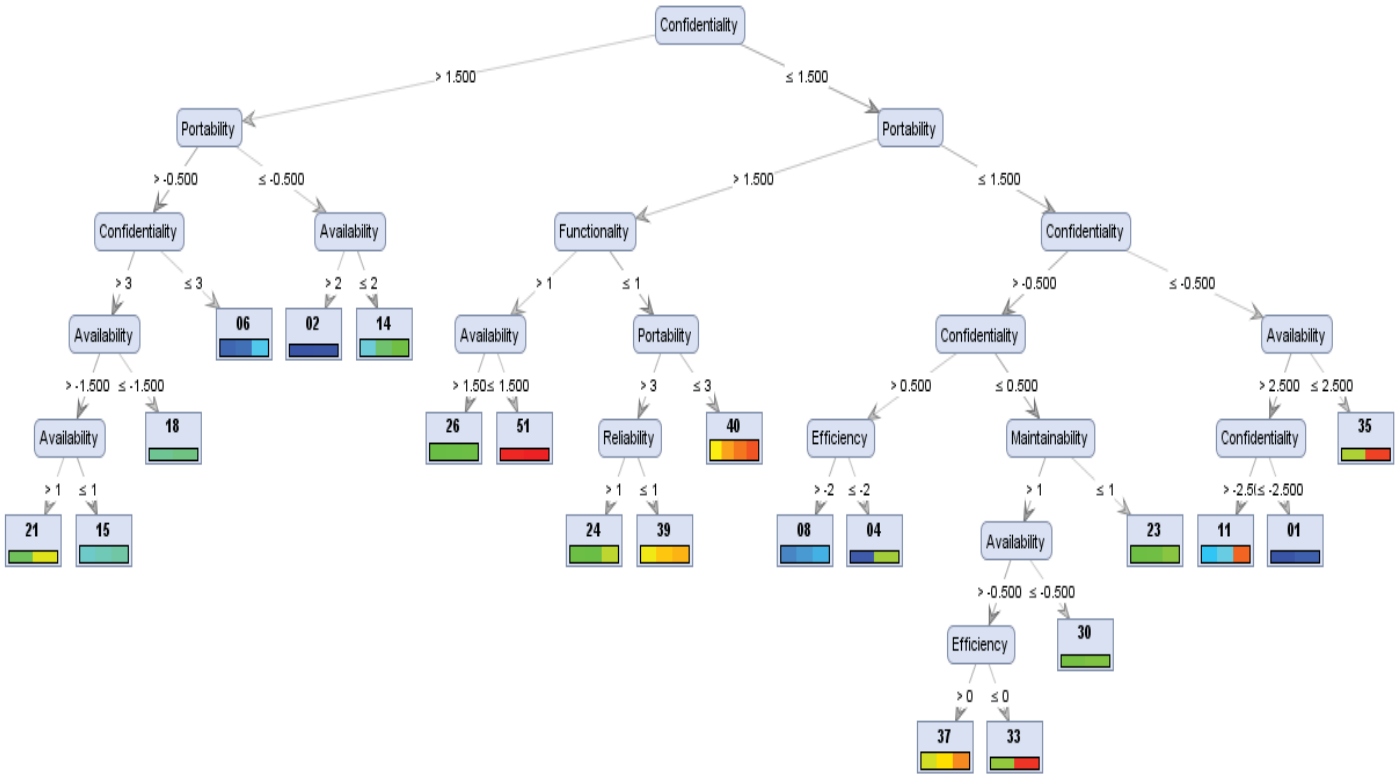


Figure 2: Decision tree for tactics selected based on specific NFR requirements levels

5 Discussion and Results

A poorly defined or incomplete Definition of Done can lead to gaps and defects in potentially shippable software as development practices vary from Story to Story. The team needs to decide whether an activity or condition belongs in the DoD for a Story, a Sprint or a Release. As we move the conditions from a story to release level, the team is temporarily creating technical debt and adding to the risk. The team should try to keep as many conditions or activities as possible at the story level and move them up to sprint or release level only if it's inefficient to do it at story level. There are many factors that will influence these decisions.

A data exported from Jira tool covering nine systems developed using Scrum methodology has been analyzed using data mining techniques (clustering and decision tree). The data size was more than 7,000 stories. Each story has 15 main attributes (dimensions). Table 5 lists and describes the attributes being preprocessed and analyzed.

Table 5: The analyzed stories data attributes (covering 7,000 stories)

Attribute (Dimension)	Description
Story ID	Identification number for each story
Story Type	Identifies the story types, whether it is Product Infrastructure, Team Infrastructure, Refactoring, Bug Fixing, or Spikes
Story	The story definition
Systems	The system identifier
Priority	The backlog priority sit by the product owner
Assignee(Scrum Team)	The Scrum team working on this story identifier
Acceptance Criteria	The main and alternative scenarios defining the acceptance criteria
Constraints	The corresponding rules and/or restrictions
Size	The story estimated size measured in story points
Sprint	Sprint identifier
Reporter	The team member delivers this story on the Jira tool
Created	Creation date
Updated	Last update date
Status	Status showing the actual progress
Versions	The associated version

Performance or load testing added to Story DoD, but if it's inefficient or expensive for the team to run performance or load testing for each Story individually, the performance or load testing can be moved from Story DoD to Sprint DoD. This allows the team to run a single performance or load testing cycle towards the end of each Sprint on multiple Stories developed in that Sprint [11, p. 4].

The injection of NFR's within the DoD's make it possible to discuss the cost of implementing non-functional requirements with business stakeholders.

Nonfunctional Requirements are persistent qualities and constraints, and, unlike functional requirements, are typically revisited as part of the "Definition of Done" for each iteration or Release (PI). NFRs exist at all three levels: Team, Program and Portfolio [8].

It is better to capture nonfunctional requirements as constraint stories. The constraint has two parts: a narrative and a list of acceptance criteria. The narrative describes the nonfunctional requirement from the perspective of the persona of the user. The criteria clarify the interaction and describe the environment. Both are required for constraint validation [5, p. 2].

6 CONCLUSIONS

The nonfunctional requirements have been explored and applying relevant qualities tactics in the early stages conducted and recommended to cover the whole product or important features in the backlog. This helps relevant stakeholders to have better user experience, and make rationale architecture and technology decisions. Whatever the chosen format to capture the non-functional requirements, it is important to capture the requirements precisely to ensure its testability.

Adding constraints to the Definition of Done increases the time required to implement each user story, as well as the time required to review or test that DoD, but it was consumed in all cases without being noticed or explicitly identified or planned.

As part of Scrum practices, it has been recommended throughout this research that incorporation of non-functional requirements should be explicitly defined by including them in the Definition of Done.

For maximizing the gained value from Scrum, adopting the engineering practices discussed is essential. This may include but not limited to TDD, CI, refactoring, etc. that can be leveraged through the injection of the discussed NFR's constraints including rules and restrictions in the DoD either at the story, or sprint, or release level.

REFERENCES

[1] R. Kartik, "Non-Functional Requirements (NFR) in Agile Practices," 2013.

[2] LeSS Company B.V., "Large Scale Scrum," 2014.

[3] V. Christophe, U. Falk and W. Brenner, "Jumpstarting Scrum with Design Thinking," 2013.

[4] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation," *Journal of Systems and Software*, vol. 83, p. 1735–1758, 2010.

[5] R. Pichler, "Discovering and Describing Nonfunctional Requirements," 2013.

[6] P. Hilton, "Scrum and non-functional requirements," Lunatech Blog, 2012.

[7] P. Loucopoulos and J. Sun, "A Systematic Classification and Analysis of NFRs," in *Proceedings of the Nineteenth Americas Conference on Information Systems*, Chicago, Illinois, 2013.

[8] Leffingwell, "Nonfunctional Requirements Abstract," Scaled Agile Inc., 2014.

[9] A. Sabry, "Decision Model for Software Architectural Tactics Selection based on Quality Attributes Requirements," in *ICCMIT*, Prague, 2015.

[10] A. Matoussi and R. Laleau, "A Survey of Non-Functional Requirements in Software Development Process," Laboratory of Algorithmics, Complexity and Logic (LACL), University Paris, Paris East, France, 2008.

[11] S. Purushotham and A. Pulla, "Bridging the Gap Between Acceptance Criteria and Definition of Done," in *PNSQC*, 2013.

APPENDIX

User Stories DoD Recommendations

A. Recommended DoD For User Stories

- Stories are created on JIRA and linked to Requirements' sheets' IDs
- A story has a clear, well-defined description
- A story lists its acceptance criteria clearly
- A story has a well-defined prototype or an existing example from a similar behavior in the system
- Integration stories are created on JIRA with a clear description
- Stories are linked to integration stories on JIRA (if applicable)
- A story's dependencies are identified and listed clearly (inter-module dependencies / intra-module dependencies)
- Code Completed and Reviewed
- Code is refactored (to support new functionality)
- Code Checked-In and Built without Error
- Unit Tests Written and Passing
- Acceptance Tests written and passed
- Pass all Non-Functional Requirements if Applicable (Cross browser compatibility tier)
- Product Owner Sign Off /Acceptance
- User Acceptance
- Manual regression scripts updated
- Test Automation Scripts Created and integrated
- Localization (truncation, wrapping, line height issues, string array issues, etc.)
- Analytics (Non-Functional Requirements) integrated and tested
- Story level device support (big browser, tablet, mobile device) tested.

B. Recommended DoD For a Sprint

- Stories meet their acceptance criteria
- Stories are implemented with compliance to their prototypes
- Stories follow the general system behavior standards
- The correct comment template is used during commits
- Integration Testing is done and bugs listed (if any)
- Test Cases and Test Scenarios are done
- The maximum number of acceptable bugs is not exceeded: Zero 'Blocker', Zero 'High', Zero 'Medium', and 'Low' : 'Enhancement or product owner decision'.
- All tracking sheets are updated (outputs of sprint)
- Unit Test Code Coverage >80%
- Passed Regression Testing
- Passed Performance Tests (Where Applicable)
- End user training team hand-off
- UAT (User Acceptance Testing)
- Production Support Knowledge Transfer done

C. Recommended DoD For a Release

- Regression tests completed in an integrated environment [11, p. 5]
- Performance or Load Tests completed
- Open defects reviewed by stakeholders and production support team
- Workarounds documented
- UAT and end user training completed

HONIC: Customer Satisfaction Index Based Framework to Transform Customer Experience

Rose Neena Tom (roseneena.tom@hcl.com)
[1]Customer Voice Measurement (CVM) Group
[2]HCL Technologies Ltd, Bangalore, Karnataka, India

SERP'15: POSITION PAPER

Abstract - *HONIC framework applies growth parameters in the transformation journey of a business relationship. A well developed and institutionalized customer satisfaction (CSAT) measurement process helps to gather business parameters or metrics. These are in-turn applied on this framework to help business to predict and proactively grow the relationship. Customer expectation and experience management procedural steps are brought in to help in the operational flow.*

Keywords: Customer Experience, Expectations, CSAT

1 Introduction

Customer experience is the product of an interaction between an organization and a customer over the duration of their relationship. This interaction includes a customer's attraction, awareness, discovery, cultivation, advocacy and purchase and use of a service [1].

Gartner defines customer experience management as "the practice of designing and reacting to customer interactions to meet or exceed customer expectations and, thus, increase customer satisfaction, loyalty and advocacy." [2]

1.1 Technology led growth

Organizations in today's world are moving away from focus on basic process improvements towards understanding capabilities of digitalization. Business led technology value chain transforms user experience effectiveness into agility, growth and profitability.

1.2 Business Transformation: Market observations

Competitive challenges posed by the advent of digital provide clear business imperatives for most organizations:

- Most companies are vulnerable to new, low-cost born-digital startups or existing competitors with strong digital strategies.
- Margins will be placed under great pressure as digital business drives down unit costs.
- End customers already expect high-end digital capabilities, and those without them will struggle to keep up[2]

1.3 Service Delivery: Transformation Program

Customer engagement is a strategic collaboration to manage, fulfill & value-add. The engagement matures along the life cycle starting from commencement, optimization, and maturity to re-invention and it is imperative to track the health of the relationship at all stages.

Transformation journey for a typical service oriented organization can be depicted as below:

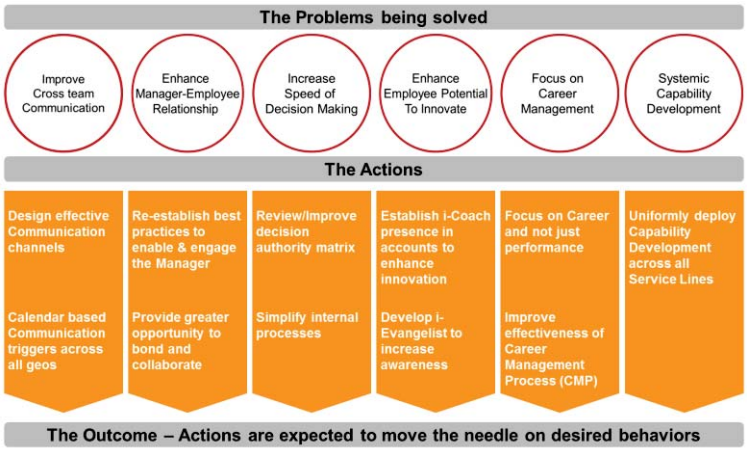


Fig. 1 Service Delivery

Digital technologies offer new opportunities to attract, engage, win, serve and retain customers. But they also add hugely to the complexity of customer strategies:

- ❖ Manage customer relationships across multiple channels and touch points
- ❖ Deliver better customer experiences across marketing, sales, service and e-commerce
- ❖ Remove the political and cultural barriers that cause customer initiatives to fail
- ❖ Unlock the real business value in your customer data

The remainder of this paper is organized as follows:

Section 2 describes the newly instituted framework with its components, objectives and methodologies. Section 3 the case-study of application of this framework.

2 HONIC Framework: Transformational Journey

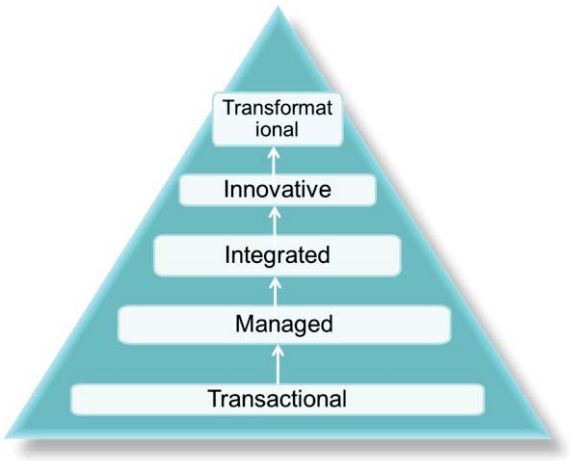


Fig. 2 H.O.N.I.C Framework

Transformational journey in a Business relationship can be found analogous with Maslow's **H**ierarchy **O**f **N**eeds – a theory in psychology proposed by Abraham Maslow in his 1943 paper "A Theory of Human Motivation" in Psychological Review [3] [4]. The hierarchy remains a very popular framework in sociology research, management training [6] and higher psychology instruction.

HONIC Framework is derived based on applying data from Business grown indicators as given in the grid below. As relationship matures over the years expectations increase and hence steps to ensure customer experience differ at each hierarchical level in the Transformational Journey.

A continuous CSAT measurement process has been deployed to gather data against each of these business parameters. Some of these measurements are carried out in monthly, quarterly or even annual cycles. Each parameter is rated on a scale 1-7; 1 being lowest and 7 being the highest. Business parameters against each of Transformational Level are developed and applied in the grid as below. In addition to these often deployed attributes additional aspects may also be applied as and when needed.

Maturity Level / Parameters								
Transformational		Integral Partner	Business Transformation	Increased revenues for clients	Client/ Corporate Governance	HCL's contribution to improving client time to market	Thought Leadership	
Innovative		Multi service approach	Industrialized Global Delivery	Showcasing value adds	Decision Making	Ability to Innovate	Alternate solutions/suppor	Innovative Approach
Integrated	Excellence in Delivery	Account Management	Predictable Delivery	Understanding clients business	Program Governance	Processes & Methodologies	Tool Deployment	Collaborating with all teams
Managed	On-Time Delivery	Project Management	Deliver right 1st time	Domain Knowledge	Project Status Update (Reports/Timelines/Accuracy)	Resource Management	Proactivness	Requirement Elicitation skills
Transactional	Responsiveness	Resourcing / Staffing	Quality of Deliverable	Technical Expertise	Communication	Soft Skills	Lowered cost of operations	Meeting SLAs

Fig. 3 HONIC: Business Parameter Grid

2.1 Framework objectives

The primary objectives of this framework are:

- Focus on customer disposition – (experience with the firm, attitudes)
- Progress on the route of being a trusted business partner
- Expectations from existing customers
- Increased wallet share
- Desire loyalty but not at the expense of lower price realization
- Advocacy through referrals

This can be achieved as depicted below:

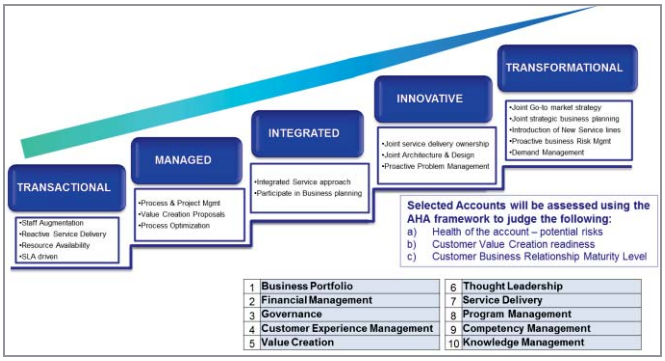


Fig. 4 Business Relationship Trajectory

2.2 Framework Institutionalization

HONIC framework was applied over the years over multiple business relationships. This has been sampled over various engagements and is believed to provide a repetitive performance and a predictable outcome.

Some of the categories of business engagements where this applied are:

- Young relationship (2-3 years)
- Growing relationships (5-6 years)
- Matured relationships (> 10 years)

3 Case-Studies

3.1 A young/budding business engagement – start of relationship

When satisfaction measurement (CSAT) parameters collated and applied for a business case study where relationship was around 2-3 years a pattern as below emerged:

3.1.1 Young business: Start of relationship

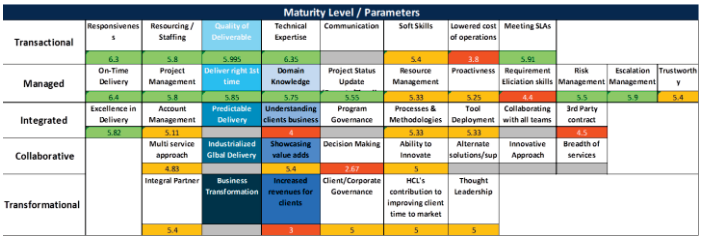


Fig 5: Young Relationship – at the start

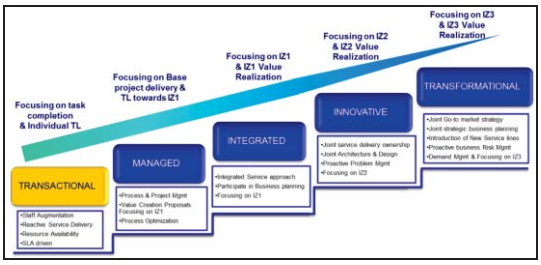


Fig 6: Young Relationship Level

3.1.2 Yong business: Subsequent Year

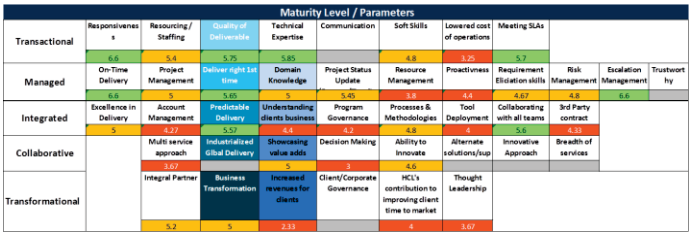


Fig 7: Young Relationship – Year 2

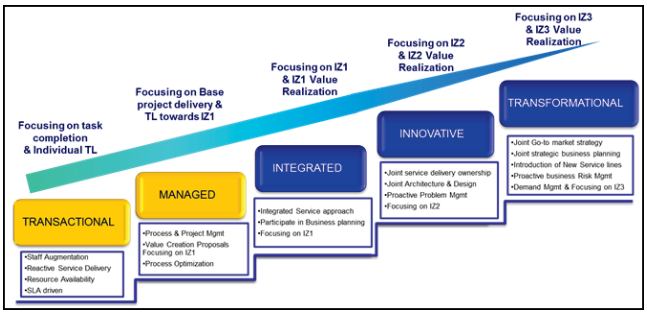


Fig 8: Young Relationship Level – Year 2

3.2 A Growing business engagement

When satisfaction measurement (CSAT) parameters collated and HONIC framework applied for a business case study where relationship was around 5-6 years the pattern that resulted is given below:

3.2.1 Growing Relationship: Year 4

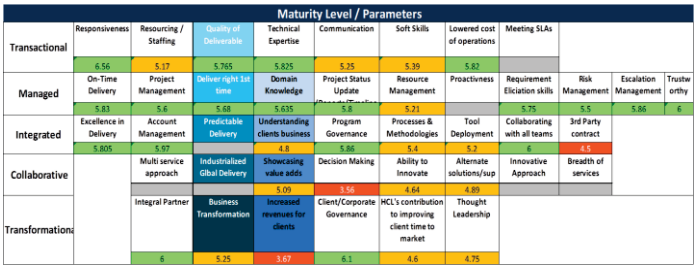


Fig 9: Growing Relationship – Year 4

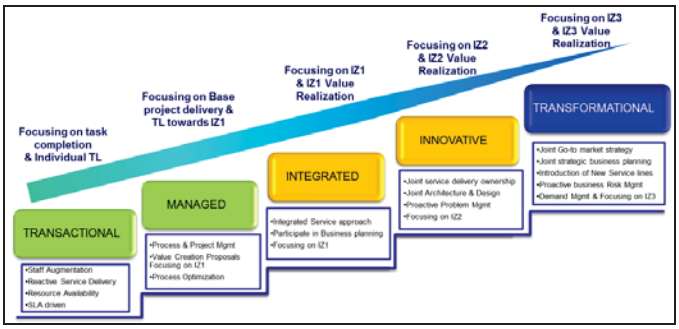


Fig 10: Growing Relationship Level– Year 4

3.2.2 Growing Relationship – Year 6

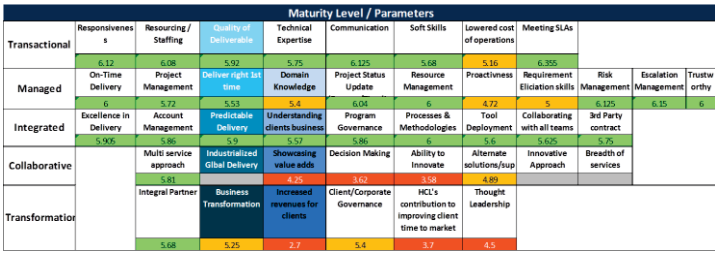


Fig 11: Growing Relationship – Year 6

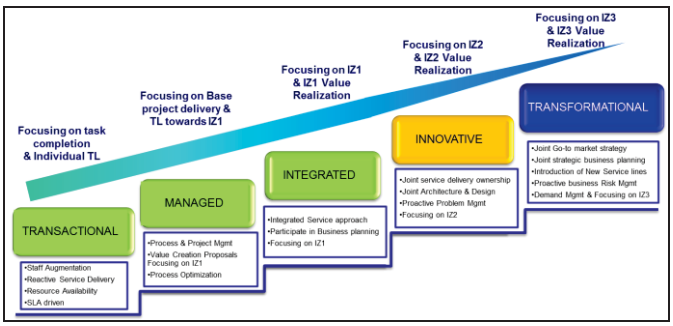


Fig 12: Growing Relationship Level– Year 6

3.3 A Mature business engagement

When satisfaction measurement (CSAT) parameters collated and HONIC framework applied for a business engagement where relationship was 10+ years the pattern that came forth was as depicted below:

3.3.1 Mature relationship: Year 12

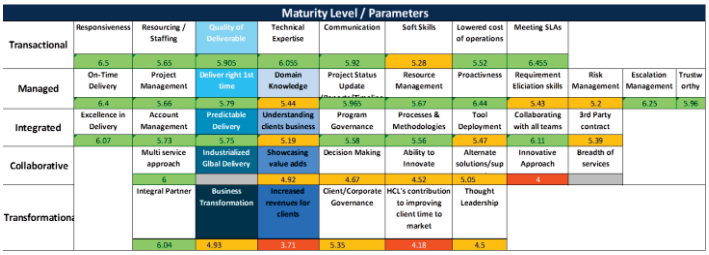


Fig 13: Mature Relationship

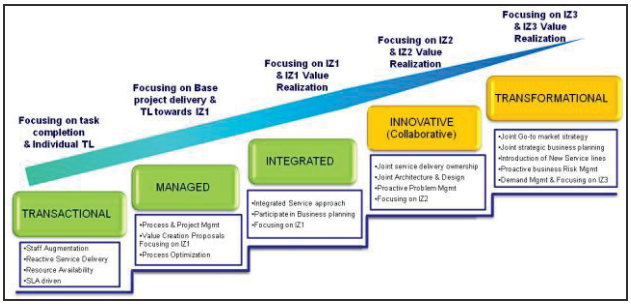


Fig 14: Mature Relationship Level

4 Case-study Inferences

Some of the salient inferences that were drawn from these outcomes are:

1. As business matures over years the relationship grows over a predicted path

2. Expectations over the years grow as per a value chain hierarchy – longer the relationship higher the expectations on the ladder

3. Businesses need to proactively find solutions and alternative approaches to cater to ever changing needs of the industry and in turn their customers

5 References

[1] http://en.wikipedia.org/wiki/Customer_experience - Retrieved 10th June 2015

[2] http://www.gartner.com/it-glossary/customer-experience-management-cem: Retrieved 13th June 2015

[3] Maslow, A.H. (1943). A theory of human motivation. *Psychological Review* 50 (4) 370–96. Retrieved from http://psychclassics.yorku.ca/Maslow/motivation.htm: Retrieved 13th June 2015

[4] https://en.wikipedia.org/wiki/Maslow's_hierarchy_of_needs : Retrieved 13th June 2015

[5] Maslow, A. (1954). *Motivation and personality.* New York, NY: Harper.

[6] Kremer, William Kremer; Hammond, Claudia (31 August 2013). "Abraham Maslow and the pyramid that beguiled business". *BBC news magazine*. Retrieved 1 September 2013

[7] Joseph P Elm, Dennis R Goldenson, 'The Effects of CMMI® on Program Performance', Carnegie Mellon® Software Engineering Institute (SEI) , 5th CMMI Technology Conference and User Group, Denver 2005

[8] CMMI® for Development, Version 1.2, CMMI_DEV V1.2, CMU/SEI-2006-TR-008 / ESC-TR-2006-008, August 2006, Carnegie Mellon® Software Engineering Institute (SEI)

[9] Robert S Kaplan & David P Norton, 'Strategy Maps' & 'The Strategy-focused Organization' & "The Balanced Scorecard – Measures That Drive Performance", *Harvard Business Review,* January- February 1992

SESSION

POSTER PAPERS

Chair(s)

TBA

Verification based Development Process for improving Reliability of Open Feedback System

JungEun Cha[1], **YoungJoon Jung, and Chaedeok Lim**
Embedded Software Platform Research Team, ETRI(Electronics Telecommunication Research Institute), Dajeon, KOREA

Abstract - *The paper relates generally to an open feedback system, which are capable of providing multiple anonymous users with content based on open source and allowing the content of multiple users to be smoothly modified and shared. Specially, the open feedback system is very important to establish a process of rapidly and accurately performing a series of processes of releasing, obtaining, and reviewing software improved by users. For this purpose, an open feedback system also needs to be systematically constructed based on the established process. We propose verification based process and software architecture for improving reliability of open feedback system.*

Keywords: Open Feedback System, Verification Process for reliability review, Open Source Verification

1 Introduction

In order to develop software desired by customers in response to suddenly changing market trends and technical requirements, a closed-form method in which only a company releases and maintains a finished software product, as in an existing method, needs to be discarded. In reality, customers desire to select and be provided with optimum hardware and software in a variety of market and functional alternative environments.

Accordingly, an open feedback system that can be evolved in various manners in accordance with users having various viewpoints based on common open source is essential to all companies, software-related communities, and personal software developers. If the development and supply of software and systems are unilaterally performed by large-sized companies, as in the past, it will be difficult for both developers and users to make mutual and continuous profits. For this reason, in order to increase companies or developers' profits by maximizing their developing abilities, they are increasingly dependent on a scheme of providing an open space in which a plurality of anonymous developers can take part and acquiring active and positive participatory opinions from the open space.[1]

An open feedback system is a system for disclosing developed content and a developed system and enabling a plurality of developers who want to use the disclosed content and system to freely share information about the disclosed content and system. That is, the open feedback system collects products processed using various technical deployment methods based on the viewpoints of a plurality of developers, reviews the collected products, and discloses the products again.

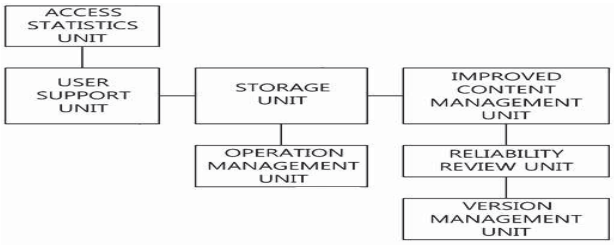
Accordingly, it is very important to establish a process of rapidly and accurately performing a series of processes of releasing, obtaining, and reviewing software improved by users. In particular, in order to sustain the diversity of products and the potential trend of development, an operator should be prevented from arbitrarily manipulating or deleting software and content related to the software. Furthermore, since an official version needs to be assigned in agreement with the opinions a majority, the feedback of users should be defined and operated in a reviewed system via an official process. The paper has been made keeping in mind the above problems occurring in the conventional art, and an object of the paper is to provide an open feedback apparatus that is capable of securing reliability and autonomy for content and a system.

2 Open Feedback System

We is defined an structure of open feedback system, including a storage unit configured to store content based on open source; an improved content management unit configured to register improved content modified based on the content by a developer with the storage unit or to update the content stored in the storage unit with the improved content; a reliability review unit configured to generate information about reliability of the improved content using a result value of reliability review performed based on the improved content; and a version management unit configured to assign a version to the improved content based on the reliability information[2].

(Figure 1) 1 is a block diagram of an open feedback system according to an embodiment of our ideas.

This work was supported by ICT R&D program of MSIP/IITP. [B0101-15-0661, The development of the self-adaptive software framework technology fo r various IoT devices]



(Figure 1) block diagram of an our open feedback system

2.1 Process for Reliability Verification

We focused the reliability verification functions to generate information about the reliability of improved content based on the result value of the reliability verification that has been performed on the improved content registered or updated. Our reliability verification part may perform the three review steps. If the previous version of content is present in the storage part, the reliability verification part may further include information about the previous version of the improved content in reliability information

A first step is to request the tester review of the improved content from a plurality of testers. In this case, the testers may refer to common users. That is, the opinions of common users who use the open feedback and whether or not an administrator review is required for a convergence result release is determined. If a plurality of testers determines that the reliability of the improved content is low, that is, if the improved content has no special advantage compared to a previous version or if there is a reason, such as the fact that an improvement is insignificant, the following review processes may not be additionally performed.

And second step for the reliability review is to request the administrator review of the improved content from the administrator In this case, if a plurality of administrators is present, that is, if collective administrators are present, the opinions of the collective administrators are collected and whether or not peer review is required is determined. As in the tester review, if the collective administrators determine that the reliability of the improved content is low, the following review processes may not be additionally performed.

The next step is the peer review. In this case, the peers may refer to the peers of a content provider who provided an open source first or the peers of a developer who have tried to distribute the improved content.

We describe the 3 steps process of performing reliability verification in the method of operating the open feedback in (Figure 2)).

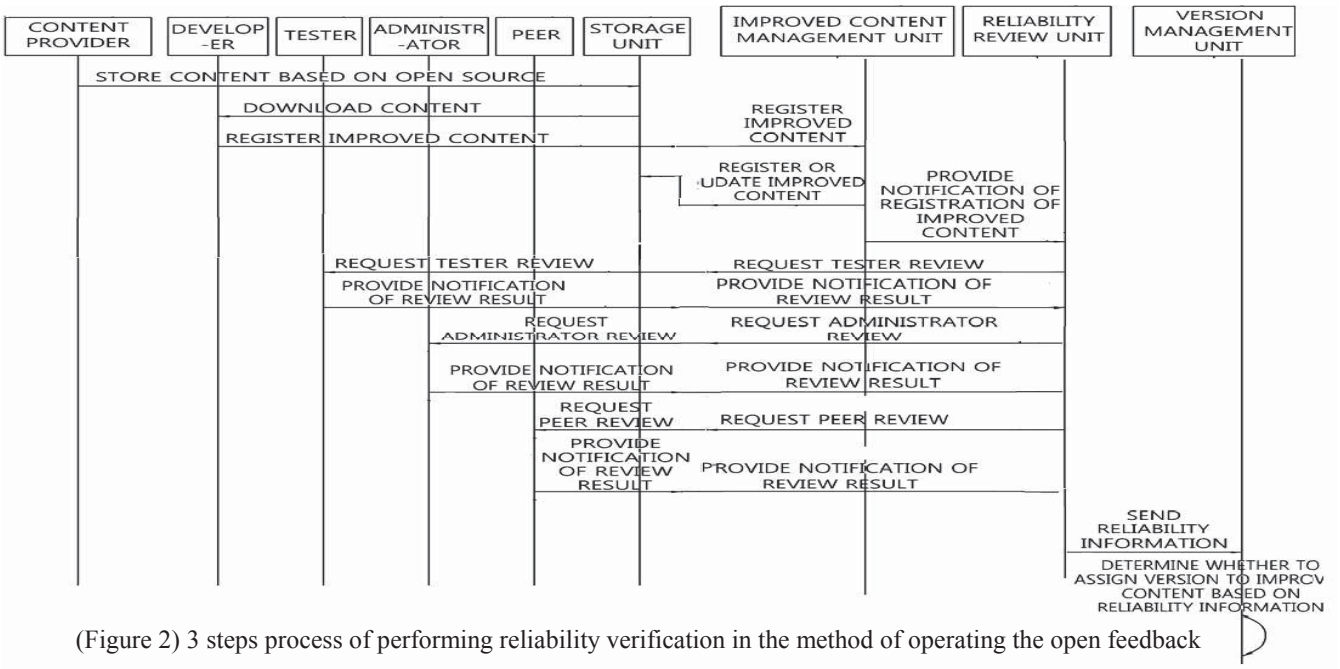
3 Conclusions

As described above, our open feedback system is advantageous in that the reliability of the development of software and content can be ensured and also the level of maintenance and repair can be improved using the collective technological intelligence of multiple users. Furthermore, it supports that content can be updated with higher quality because a plurality of reliability review procedures is performed upon updating the versions of content distributed by a content provider or a user. Currently, we are constructing and maintaining the feedback system(www.opensw-seed.org) applied our verification process.

4 References

[1] JKarl Popp,Ralf Meyer, Profit from Software Ecosystems, Books On Demand, Act 2010

[2] Jung-Eun cha, YoungJun Jung, ChaeDeok Lim, "Model Based Design method for Dynamic Configurable Ecosystem of Embedded System", 11th ESA 2013, pp81-86, July 2013.



(Figure 2) 3 steps process of performing reliability verification in the method of operating the open feedback

SESSION

LATE BREAKING PAPERS: SOFTWARE ENGINEERING PROCESSES + SECURITY ISSUES

Chair(s)

TBA

Semi-supervised learning applied to performance indicators in software engineering processes

Leandro Bodo[1,3],
Hilda Carvalho de Oliveira[1,3], Fabricio Aparecido Breve[1,3], Danilo Medeiros Eler[2,4]
[1]Dep. of Statistics, Applied Mathematics and Computer Science
[2]Dep. of Mathematics and Computer Science

Universidade Estadual Paulista, UNESP
[3]Rio Claro, [4]Presidente Prudente - Brazil
lbodo@rc.unesp.br, hildaz@rc.unesp.br, fabricio@rc.unesp.br, daniloeler@fct.unesp.br

***Abstract*—Performance indicators are critical resources for quality control in the software development process. Over time, the data volume of the historical basis these indicators increase significantly. Moreover, the diversity of treatment (individual or group) and the frequency of data collection hamper the analysis. The time and reliability of these analyzes are important to support a faster and more effective decision. Thus, this paper proposes the use of artificial neural networks with semi-supervised learning to analyze the historical basis of performance indicators. In order to support the sample labeling process it is recommended to use information visualization techniques. An indicator reference model was defined based on the software process reference model MPS for Software (MPS-SW) to be used before the labeling process.**

***Keywords*—Software Processes, performance indicators, machine learning, artificial neural network, MPS-SW.**

I. INTRODUCTION

During a software development, procedures for quality control aim to identify ways to mitigate causes of unsatisfactory results. Furthermore, these procedures also allow to evaluate the performance of the processes and to indicate the need for changes and corrections on the process [1], [2]. Specific project results must be monitored to determine if they are in compliance with the quality criteria previously defined and which are relevant to the development of the software. Statistical techniques have helped the evaluation of the results of quality control.

The performance indicators are important tools to the quality control in general. These indicators aid to quantify the performance of activities, processes and products, making it possible to analyze the results and compare them to the planned goals [3]. The indicators provide numerical relations that reflect the current state of the processes. These relations provide information for decision-making related to the processes, and consequently, to their own business. The performance indicators can present variations that require the management's attention when making a decision. It is possible to evaluate the variations that occurred and generate prognostic (projections) through historical comparisons of an indicator.

The performance indicators might be analyzed either individually or in groups, depending on the defined specifications to support the decision-making. These groupings can also consider distinct indicators from different projects or products, or consider the same indicator that comes from different projects or products. The analyses are performed continuously during the software development process, in order to create a historical basis and judge the quality of the software during the whole process [2].

However, the historical basis of the performance indicators tends to become large and complex That happens because of the large amount of data that is stored at the same time and the intrinsic diversity of the indicators (different types, granularity and frequency of data collection). Furthermore, the data volume that is produced by these indicators tends to increase dramatically over the monitoring time. All these factors require a solution that enables the analysis of performance indicators, improving the use of these indicators in the software development process.

Within this context, this paper proposes the use of semi-supervised machine learning techniques for analysis of performance indicators in software development processes. This type of learning reduces the cost with labeling process of supervised learning and does not despise the labels of samples, it could happen in the unsupervised learning.

The goal is to "teach" (to train) an Artificial Neural Network (ANN) to recognize the existing patterns in the large volume of the historical data generated by the indicators. Thus, the ANN will be able to analyze the indicators in different development processes and provide results that indicate the status of a particular situation, similar to a traffic light. Overall, an ANN can analyze indicator groups simultaneously, with different complexities. This allows the indicator groups to be controlled in dashboards. The same goes for individual indicators. The same goes for individual indicators. This technique is able to provide a decision-making more effective

and faster.

Therefore, the *section II* presents an overview of machine learning and ANN. The *section II* outlines the semi-supervised learning algorithm for analysis of performance indicators: Particle Competition and Cooperation (PCC).

ANNs have been used for treatment of performance indicators in different applications of Software Engineering. The section *IV* presents a few comments about it and discusses some criteria for the use of ANNs in software development processes.

On the other hand, the *section V* outlines an indicator model based on processes levels G and F of reference model MPS for software (MPS-SW). This model will help to guide the grouping of indicators in software development organizations.

The *section VI* shows the application of the PCC algorithm on a real set of performance indicator samples. These indicators belong to the historical basis of a company that develops software, that is a project partnership and certified in the level G on the model MPS-SW. The results of PCC application are compared with the results that have been obtained in two supervised ANN techniques. There was no comparison with unsupervised methods, because this type of learning ignores the information of the sample labels - and this information is fundamental for the type of problems presented. The final considerations of this article are presented in *section VII*.

II. Machine Learning and Artificial Neural Network

Within the context of artificial intelligence, machine learning is an area that involves the construction of systems capable of acquiring knowledge automatically. This area considers the development of algorithms that use the acquired "experience" to produce results without human intervention. A machine learning algorithm can make decisions based on examples of input data [4].

Overall, machine learning algorithms require the analysis of a large amount of samples for learning. The idea is to teach the algorithms to solve different problems in a given context. This context may have characteristics that cause changes over time and/or the type of application and use [5].

ANN is a type of machine learning techniques, which is discussed in *subsection A*. There are different machine learning categories, each one recommended for a particular type of problem. The *subsection B* presents three categories: supervised, unsupervised and semi-supervised.

A. Artificial Neural Network (ANN)

ANNs are computational techniques based on mathematical models inspired by the neural structure of intelligent organisms. The acquisition of knowledge in ANNs is performed through experience [6]. They have the ability to learn by examples, making inferences from that learning and improving their performance gradually.

ANN has a behavior based on groups of neurons in the human brain, which receive and transmit information through the dendrites and axons, respectively [7]. When a specific set of data inputs and their outputs are presented with an ANN, it is able to auto adjust its synaptic weights. The adjustment of the connections is obtained by learning adopting as training criterion (and subsequent analysis) a specific activation function. The training phase of ANNs consists in a functional relationship mapping that exists between the inputs and outputs. Following training, the network should be able to generalize the behavior of the process at the time when other inputs are presented to it (different inputs from those used during training) [8].

ANNs can be used for performance evaluation indicator groups of software engineering processes, as proposed in this paper. Indicators are considered the network attributes and have different measurement value and goals, without a pattern in data types defined (may contain integer, boolean or real values). Therefore, a software factory can "teach" how certain indicator groups can express control targets, adjusting its parameters on demand.

B. Machine Learning categories

A supervised learning algorithm requires that an expert (external entity) introduce some sets of patterns for the inputs and the corresponding standards for the outputs (results). An output can be a numeric value or can predict a class label for the input object. In the training phase of an ANN, for example, the expert indicates explicitly for each input if the output response is good or bad (data labeling process). Thus, the result provided by the network is compared to the expected answer. If the result is different than expected, an error is reported to the network and adjustments need to be made in order to improve future responses.

Unsupervised learning algorithms do not require an external entity to perform the training process. They aim to determine how the data is organized only based on the input patterns, without labels or output values. These algorithms process the inputs available and try to establish internal representations to encode features and classify them automatically, by detecting the singularity in the input samples.

Semi-supervised learning algorithms use both labeled and unlabeled data for training. In many cases, the use of few labeled data with many unlabeled data considerably improves accuracy of the learning [9].

Due to the large amount of existing databases, label data for supervised learning algorithms have become an increasingly unworkable process. Usually, the labeling process is expensive and time consuming, requiring intense involvement of human experts. On the other side, the unsupervised algorithms ignore valuable information label of the data items. Semi-supervised algorithms can mitigate these problems: a few labeled data items are combined with a large amount of unlabeled data, producing better classifiers [10].

III. Algorithm ANN semi-supervised adopted

The selected algorithm to propose a treatment for performance indicators in this paper is called Particle Competition and Cooperation (PCC) [9].This semi-supervised ANN algorithm was chosen because it requires little human effort and consequently little financial cost.

The PCC algorithm consists of a graph of related networks, which has various particles moving through the network. These particles are organized in the form of teams, where particles of the same team go over the network in a cooperative way in order to propagate their labels. Meanwhile, particles of different teams compete with each other to determine the borders of the class, and reject intrusive particles.

Traditionally, labels are spread over the network in a global way in other semi-supervised learning algorithms based on graphs. This means that the information is propagated from all nodes to all other nodes for each iteration of the algorithm, accordingly to the respective edge weights. On the other hand, the spread of the label occurs locally at the PCC algorithm. Therefore, each step of the algorithm, every particle propagates its label to a selected neighbor by the rule "random[1]-greedy[2]". Thus, each particle visits only a portion of the nodes that potentially belongs to its team (subnet), preventing from redundant operations are carried out [9].

Breve [10] presents a metaphor of particles based on ant behavior (**Fig. 1**) to explain the PCC algorithm.

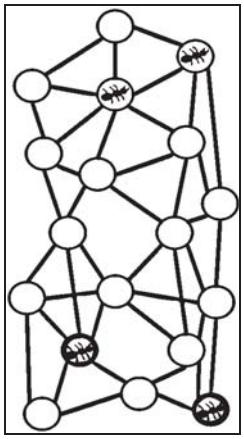


Fig 1. Metaphor based on behavior ant [10].

Every node on the network has an array of elements, responsible for representing a level of particles each team. Initially, the input data set is transformed into a non weights and undirected network, where each node corresponding to a non-labeled sample will have its field level configured with the same value $\frac{1}{c}$, where c = number of classes / teams. The **Fig. 2 (a)** explains the initial situation with four classes, ie, nodes not labeled with domain levels 0.25: [0.25 0.25 0.25 0.25]. Next, a set of ants are placed on the network. Each of them is a labeled data item that is set to the highest value. The **Fig. 2 (b)** illustrates this situation to the four classes labeled as Class A: [1 0 0 0]. The subset of particles with the same label is called "team" [9].

1*Random walk*: each particle chooses any neighbor to visit at random, without worrying about the domination levels or distance from it home node. It is a movement for acquisition of new node and exploration [9].

2*Greedy walk*: each particle visits the nodes that are closest to their home nodes, especially those who are already dominated by its own team. It is a movement to defend the territory of his team [9].

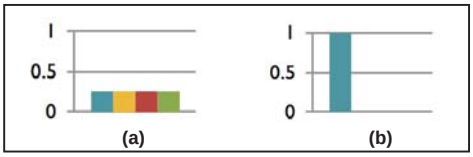


Fig. 2. Initial domination level: (a) unlabeled sample; (b) labeled sample.

Every ant chooses a neighboring node to visit each iteration of the algorithm, using the "random-greedy" rule. The domain level of their team is increased when an ant selects a neighboring node. Meanwhile, domain levels of other teams are decreased. If the node to be visited is in control of its own team, the ant gets stronger, increasing their domain levels. However, the ant weakens if the visited node is domain of the other team. Each ant works prioritizing the domain of their respective neighborhoods (neighboring nodes). For this, they have a particular node as home and each ant keeps information of the distance between their respective homes and other network nodes. This method includes cooperation between the ants. Thus they prioritize helping their teammates with their neighborhoods, and eventually try to invade opponents' territories. Each team tries to dominate the largest possible number of nodes, and simultaneously try to avoid the invasion of ants from other teams. At the end of the iterative process, each unlabeled data item will be labeled according to the team's label which contains the highest domain level.

IV. Artificial Neural Network applied to performance indicators

Machine learning techniques have been used with performance indicators in different areas of knowledge. Melo et al. [11] used an ANN Multilayer Perceptron (MLP) to predict the variation in the flow of vehicles on roads, helping drivers to selecting the best routes. Neto, Nagano and Moraes [12] used an unsupervised ANN to classify agricultural cooperatives based on their socioeconomic indicators. Cattinelli et al. [13] used an ANN to analyze groups of performance indicators in 109 hemodialysis clinics in Italy, Portugal and Turkey. Within the context of Software Engineering, Kutlubay et al. [14] used ANN techniques for detecting defects in software products.

In this way, this paper proposes the use of PPC algorithm (see *section III*) for an ANN with semi-supervised learning for quality control during software development. This proposal has a cost of labeling data up to 20% - which is considered low cost. However, it will allow the development of tools that will automate the monitoring processes effectively and efficiently. The reliability of the labeling process can be aided with information visualization techniques. It is recommended the LSP techniques (Least Squares Projection) and parallel coordinates.

Overall, the software processes and the performance indicators are defined for any different software developer organization. Thus, an indicator model was created to guide the grouping of performance indicators before using an ANN. This

model will serve as a reference to the initial work of selection and categorization of the indicators in the organization before the labeling process, as shown in *section V*.

V. Software Process Reference Model for Performance Indicators

As software organizations have specific indicators and processes, the processes included in the MPS-SW quality model were considered as reference for this work. Initially, seven processes are being considered as levels G and F models. The largest numbers of certified software organizations are in these two maturity levels (almost 90% of the certifications). An overview of this model is presented in the *subsection A*.

The *subsection B* presents an indicator model based on the processes of the maturity levels G and F of the MPS-SW model, to support the selection and clustering of indicators before the application of the ANN. The company's real indicators may be converted to the MPS-SW model indicator. However, not all model indicators can be converted into real indicators.

A. MPS-SW model

The software MPS-SW model is part of Brazilian Software Process Improvement Program (MPS.BR) created in 2003. This model is based on ISO / IEC 12207 (software lifecycle) and ISO / IEC 15504 (software evaluation). Currently there is an agreement between the Software Engineering Institute (SEI) and the Association for Promoting the Brazilian Software Excellence (SOFTEX) for joint assessment and certification of MPS-SW models and CMMI-DEV [17]. The MPS-SW model is designed to benefit mainly micro, small and medium software enterprises (MSME).

The MPS-SW model has seven maturity levels aiming a gradual implementation and certification from the first level: G. This maturity level includes two critical software processes for MSME: Requirements Management and Project Management. On each level are added new processes, as shown in **TABLE I**. This means that a higher level involves its processes more the processes from the lower levels. The highest level, A, involves all processes from the lower levels and emphasizes the continuity of the improvement in processes [18].

TABLE I. Processes added to each Maturity Level (ML) of the MPS-SW [18].

ML	PROCESSES
A	(*no new processes are added*)
B	Project Management (*new outcomes*)
C	Decision Management; Risk Management; Development for Reuse
D	Requirements Development; Product Design and Construction; Product Integration; Verification and Validation
E	Human Resources Management; Process Establishment; Process Assessment and Improvement; Project Management (*new outcomes*); Reuse Management
F	Measurement; Configuration Management; Acquisition; Quality Assurance; Project Portfolio Management
G	Requirements Management; Project Management

The utilization of performance indicators is required for the process from level F - that is kept up to level A. However, a good practice is to adopt performance indicators from level G.

B. Reference model for indicators

A business ontology was developed by Pizzoleto [16] organizing the levels G and F of the MPS-SW model. This ontology proposed performance indicators for almost all the expected results in the processes. **Fig. 3** shows the Protégé system screen with part of the measurements process of the F level, as described in the ontology.

Using this ontology, interviews in MPS-SW certified software companies were held. Based on literatures on performance indicators in Software Engineering, such as [19], [20], [21] and [22], an indicator model has been developed for the processes of the levels G and F. **Fig. 4** shows the indicator model with the following attributes: description, purpose, calculation method, measure unit, collection frequency, results presentation frequency and scope application (project, product and business).

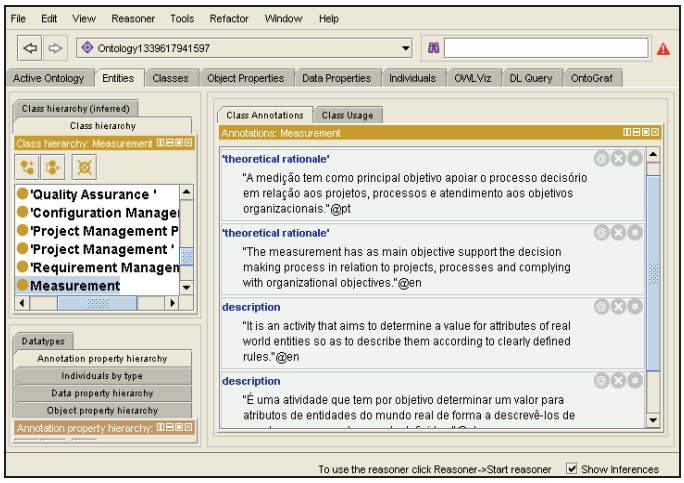


Fig.3. Measurement process from the MPS-SW ontology in the Protégé system [19].

Level	MPS-SW PROCESS	Description	Goal	Calculation	Application
	GQA4	Variation of planning	Shows the percentage of variation on sprint project planning	Total implementation of requirements planned on project - Total requirements implemented on project) * 100	Project
F	GRE1	Open requirements	Evaluate the number of requirements which were open on day	Sum of total open requirements which were open on day	Product
	GRE5	Impediment requirements	Shows the percentage of requirements responsible for project offside	(Total Impediment requirements / Total open requirements early in the project) * 100	Project
G	GRE4	Deployed requirements	Shows the number of requirements that have been implemented on day	Sum of all the implemented requirements on day	Product
	GPR9	Project Planning	Present the percentage of total time the project coordinator acted with planning	(Total hours that pointed only at the project planning / Total hours pointed out by project coordinator) * 100	Project

Fig. 4. Representation of the reference model of performance indicators.

A group indicator model was defined based on four perspectives of the Balanced Scorecard strategy (BSC): Financial, Customer, Internal business processes, and Learning

and growth. These perspectives reflect the vision and organizational strategy of a company [23]. The proposal was to associate the indicators used in the model presented in **Fig. 4** with the categories of BSC perspectives. **Fig. 5** illustrates some of these indicators associated with each view. This model supports the software organization to form sets of indicators to the process of labeling and training of ANN. The output result for each perspective will be the "status" of the analyzed set of indicators.

BSC	INDICATOR	TRAFIC LIGHT
Financial	...	
	Rework during implementation of the requirement	
	Efficiency of project planning	
	Impediment requirements	
	...	
Customer	...	
	Customer service policy	
	Number of notifications generated by customers	
	Cumulative requirements	
	...	
Internal business processes	...	
	Variation of planned	
	Project planning	
	Adherence to the processes	
	...	
Learning and growth	...	
	Training of programmers	
	Training of analysts	
	Training of project team	
	...	

Fig. 5. Indicators groups based on the BSC perspectives with three possible outputs.

VI. RESULTS

In order to evaluate the use of ANNs on analysis of performance indicators in software processes, experiments were performed with two different types of learning: supervised and unsupervised. Two algorithms of ANN supervised that are well-known in the literature were selected: Multilayer Perceptron (MLP) and K-Nearest Neighbor (KNN). The intention was to compare the results with the PCC algorithm, presented in *section III*.

A real indicators database used in software processes was applied as an input to the ANN. These indicators were chosen from the reference model of the proposed indicators in *section V*. The data were obtained from a company certified in level G of MPS-SW model. This company, a partner of the project, develops software for public management. The *subsection A* provides more information about the database used.

The output classification was made using very simple criteria, similar to a traffic light, as shown in **Fig. 6**. In this metaphor, the green light indicates that it is to continue the process execution because the results of the analysis in the indicators group report that the situation is satisfactory. The yellow warning signs to pay attention, because the level of satisfaction at the previously set target is regular. Already the red light indicates that the process should be stopped, to be unsatisfactory - too far from the established pattern.

Thus for ANNs, the "Green" label obtained "satisfactory", the "Yellow" label "Regular" and the "Red" label "Unsatisfactory". Therefore, if the output provided by the ANN obtained labeled "satisfactory" (Green), then the group of the indicators is analyzed in accordance with the desired - the processes are controlled. If the result of the output is "Regular" (Yellow), then the process has breaks, so greater attention is needed in case management. Finally, if the output is "unsatisfactory", the processes are not effective, requiring corrective actions.

Although the applications of the results of ANNs algorithms are presented in *subsection C*, some relevant information about the experimental procedure is presented in *subsection B*.

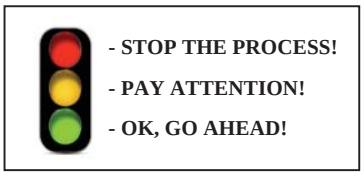


Fig.6. Traffic light metaphor to classification of ANN.

A. Indicators database used

The company that provided the performance indicator data has been working in software production for over 30 years and has hundreds of municipal governments as clients. The historical basis includes indicators of four years ago. This paper does not include the process of the data collection, but it is worth remembering that the MPS-SW certification (level G) provides some guarantees of the use of the best practices and data reliability.

The **Fig. 7** presents the statistical data related with the database used, considering three projects, identified as "A", "B" and "C". It is worth mentioning that the company's project managers collaborated with the labeling process of the samples:

- Number of instances: 300;
- Number of attributes: 3;
- Missing values: none;
- Distribution of grades: 33% for each one of the classes;
- Information about the attributes (indicators):
  a) Open requirements per daily in Project A;
  b) Open requirements per daily in Project B;
  c) Open requirements per daily in Project C;
  d) Cluster:
    - Cluster 1: Satisfactory;
    - Cluster 2: Regular;
    - Cluster 3: Unsatisfactory.

DESCRIPTION	MIN	MAX	AVG	STDEV
Project A	5	80	28.55	13.61
Project B	1	76	29.66	11.00
Project C	4	70	24.86	11.00

Fig. 7. Statistical data related on the database of the indicators used.

B. *Information about the experimental process*

Before presenting the results of the ANN techniques, it is important to present some relevant information and certain criteria adopted.

First of all, the software tool used for the application of supervised learning algorithms is called Weka, from the University of Waikato.

The Matlab system was used to implement the PCC algorithm proposed for analysis of the performance indicators (*section III*).

The time spent by the humans in labeling process of the sample was calculated from the sum of each measurement performed. This time will be important in the experiments presented in *subsection C*. Already the time spent in the review process made by the algorithms was measured by specific functions in each tool.

It is worth noting that each attribute had its normalized value before being used in the algorithms.

The training was made using ten different configurations, according to the percentage of trained sample: 2%, 4%, 6%, 8%, ..., 20%. Thus, each algorithm was executed ten times for each trained percentage, and the result presented is the average of the ten runs. This was necessary because the algorithms used are stochastic pattern and the data labeled in training were random.

C. *Application of Artificial Neural Network algorithms*

The grouping of indicators was based on the reference model presented in *section V*. The data from the same indicator were grouped but relating to three different projects (A, B and C). This kind of grouping allows a comprehensive analysis of the progress of different projects and/or the company's products, in relation to the selected requirement.

As presented in *subsection A*, 300 instances (100 of each cluster) and three attributes were used. As shown in *subsection B*, training of ANNs was carried out gradually from 2% to 2% by selecting that percentage of samples for training and the rest for validation.

The **Fig. 8** shows the result of the accuracy rate for each algorithm, according the increase in the percentage of trained samples. It can be noticed that the KNN algorithm presented good results with 2% trained samples, but the result got close to 81% accuracy when 20% of the samples were trained. The MLP algorithm achieved strong growth at the beginning and gradual growth until the end, almost reaching 85% accuracy. The PCC algorithm achieved about 76% accuracy with 2% of the trained samples. It had an increase of 6% with 4% of the trained samples and gradually increased from 82% to 84%, with 12% of the trained samples. Considering the range between 12% and 20% of trained samples, the PCC algorithm showed a significant increase of 84% to 92%. So, the best accuracy rate was achieved with the PCC algorithm for all variations of training, even with few data for training.

In relation to the accuracy by class, the **Fig. 9** shows the result of better time PCC algorithm. The algorithm correctly classified 96 samples as satisfactory and missed four, classifying them as Regular. In Cluster 2 (Regular) the PCC algorithm correctly classified 86 samples and 14 wrong, and 2 as satisfactory and 12 as cluster 3 (Unsatisfactory). The cluster 3 obtained 86 correct samples and missed 14, classifying it as Regular.

Another important point is the performance of PCC algorithm for analysis of performance indicators compared to a human (HUM). The **Fig. 10** shows that as the amount of data increases the difficulty of the human being also increases significantly. The measurement obtained by the analysis of a human being had an expert in the field during the data labeling step. This specialist visually analyzed the indicator results on each project and compared with its expected result.

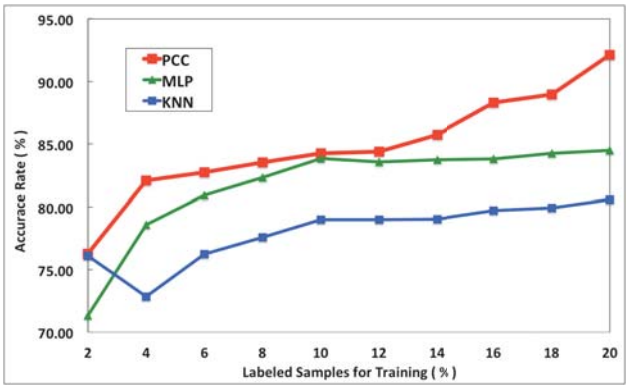


Fig. 8.Accuracy rate by technique.

		PREDICTION		
		Satisfatory	Regular	Unsatisfactory
ACTUAL	Satisfatory	**96**	4	0
	Regular	2	**86**	12
	Unsatisfactory	0	14	**86**

Fig. 9. Accuracy rate with cluster: actual vs prediction.

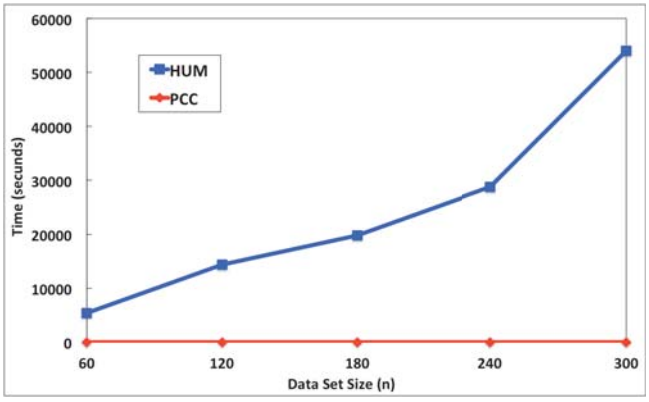


Fig10. Runtime: algorithm PC vs human (HUM).

VII. Conclusions

This paper proposes the use of semi-supervised ANN technique called Cooperation and Competition between particles (PCC) to support the performance indicator analysis

in processes of software engineering. Results with real data bases showed that the PCC algorithm achieved excellent accuracy rate, higher than the two supervised learning algorithms used.

The use of an ANN allows composed groupings indicators of different processes, products and designs and enables analysis of individual indicators. Tools that implement ANNs provide a differentiated overview of what is being measured, regardless of the complexity of the data. This contributes to the use of performance indicators in software development companies with different sizes, including micro, small and medium enterprises. Reference models proposed for selection and grouping of indicators contribute to assist organizations in selecting the appropriate settings to quality control processes.

REFERENCES

[1] PMI - Project Management Institute INC (Pennsylvania). Um guia do conhecimento em gerenciamento de projetos: Guia PMBOK. 4. ed. Newtown Square: PMI, 2008. 459 p.

[2] I. Sommerville, Engenharia de Software. 9. ed. São Paulo: Pearson, 2011. 529 p. Tradução de: Kalinka Oliveira; Ivan Bosnic.

[3] L. H. Boyd, J. F. A. Cox, "Cause-and-effect approach to analyzing performance measures". Production and Inventory Management Journal, v. 38, n. 3, p. 25-32, 1997.

[4] T. Mitchell, "Machine Learning". [s.i]: McGraw Hill, 1997.

[5] E. Alpaydin, "Introduction to machine learning". Cambridge, Ma: The MIT Press, 2004.

[6] S. Haykin. Neural Networks: a comprehensive foundation. Upper Saddle River, Nj: Prentice Hall, 1994. 768 p.

[7] Z. Kovacs, Redes Neurais Artificiais: Fundamentos e Aplicações. São Paulo: Edição Acadêmica, 1996.

[8] A. P. Braga, A. C. P. L. F. Carvalho, T. B. Ludemir, Redes Neurais Artificiais: teoria e aplicações. Rio de Janeiro: LTC, 2000.

[9] F. A Brave, L. Zhao, M. G. Quiles, W. Pedrycz, J. Liu, "Particle competition and cooperation in networks for semi-supervised learning". IEEE transactions on knowledge and data engineering, [s.i], 2009.

[10] F. A. Breve, Aprendizado de máquina utilizando dinâmica espaço-temporal em redes complexas. 2010. 165 f. Tese (Doutorado) - Curso de Ciências de Computação e Matemática Computacional, Departamento de ICMC-USP, Universidade de São Paulo, São Carlos, 2010.

[11] R. G. Mello, D. A. P. Junior, J. F. G., Oliveira, C. F. Bremer, Avaliação de desempenho para o gerenciamento estratégico do chão de fábrica. ANPAD. 14p, 2000.

[12] S.B. Neto, M. S. Nagano, M. B. C. Moraes, Utilização de redes neurais artificiais para avaliação socioeconômica: uma aplicação em cooperativas. R. Adm, São Paulo, v.41, n.1, p.59-68, 2006.

[13] I. Cattinelli, E. Bolzoni, M. Chermisi, F. Bellocchio, C. Barbieri, F. Mari, C. Amato, M. Menzer, A. Stopper, E. Gatti. Computational intelligence for the Balanced Scorecard: Studying performance trends of hemodialysis clinics. Artificial Intelligence in Medicine archive, v 58, I3, p. 165-173, 2013.

[14] O. Kutlubay, M. Balman, D. Gül, A. B. Bener, A Machine Learning Based Model for Software Defect Prediction, working paper, Bogazici University, Computer Engineering Department, 2005.

[15] SOFTEX - Associação para Promoção da Excelência do Software Brasileiro. MPS. BR Melhoria de processo do software brasileiro: Guia de Implementação – Parte 9: Implementação do MR-MPS em organizações do tipo Fábrica de Software, 2012a. Available at: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_Parte_9_20111.pdf>. Acesso em: 01 jun. 2015.

[16] A. V. Pizzoleto, Ontologia Empresarial no modelo MPS.BR visando modelagem de processos de negócios, com foco nos níveis G e F. Dissertação (Mestrado) – Universidade Estadual Júlio de Mesquita Filho, 2013.

[17] M. Kalinowski, G. Santos, S. Reinehr, M. Montoni, A.R. Rocha, K.C. Weber, G.H. Travassos, "MPS.BR: promovendo a adoção de boas práticas de engenharia de software pela indústria brasileira". XIII Congreso Ibero americano en "Software Engineering" (CIBSE), Cuenca, Ecuador, 2010.

[18] M. Kalinowski, K. Weber, N. Franco, E. Barroso, V. Duarte, D. Zanetti, G. Santos, "Results of 10 Years of Software Process Improvement in Brazil Based on the MPS-SW Model". 9th International Conference on the Quality of Information and Communications Technology, IEEE, 2014.

[19] A. V. Pizzoleto, H. C. Oliveira, Methodology for ontology development in support to the MPS model for software. In: International Conference on Software Engineering Research And Practice (Serp), 11. 2013, Las Vegas-Nevada, 7p, 2013.

[20] T. R. Ojha, Analysis of hey performance indicators in software development. Master on Science Thesis. Tampere University of Technology, 2014.

[21] G. Santos, M. Montoni, R. C. S. Filho, A. E. Katsurayama, D. Zanetti, A. O. S. Barreto, A. R. Rocha, Indicadores da Implementação do Nível E do MR-MPS em uma Instituição de Pesquisa. VIII Simpósio Brasileiro de Qualidade de Software.

[22] R. T. Moreira, G. N. Lima, B. B. Machado, W. T. Marinho, A. Vasconcelos, A. C. Rouiller, Uma abordagem para melhoria do processo de software baseada em medição. VIII Simpósio Brasileiro de Qualidade de Software.

Big Data: Security Engineering Framework

A. Daya Gupta[1], **B. Shruti Jaiswal**[2], **and C. Prudence Kadebu**[3]
[1,2] Computer Science and Engineering, Delhi Technological University, Delhi, India
[3]Software Engineering, Harare Institute of Technology, P O Box BE277 Belvedere, Harare, Zimbabwe

Abstract - *The area of Security Requirements Engineering is becoming one of the most widely researched areas in Software Engineering, having captured the interest of the Software Engineering community for its profound importance in the development of robust and secure software. However, the maturity of Security Requirements Engineering as applied to Big Data is still in its infancy stage as shown by scarcity of related material in literature. Big Data refer to the recent growth in data generated by technologies such as social networks at a very high rate with qualities of high volume, high velocity and variety. Big Data has brought many security concerns, as far as protecting data in these highly distributed environments is concerned. These emerging technologies have not only made data management more flexible, but have also increased the attack surface by introducing many vulnerabilities making data very much at risk from various threats. Recent studies have shown that the design of the Big Data stores was meant to improve scalability, performance and flexibility while security was given less priority. Hence, it is imperative to ensure that Security Requirements Engineering for Big Data environment is given a profound consideration so as to protect sensitive information stored therein. In this paper a framework for Security Requirements Engineering for Big Data is proposed. Security Requirements are elicited from vulnerabilities inherent in the Big Data stores based on generic operations (Create, Read, Update, and Delete) performed on the database. If the database is secure then fewer resources needs to be invested in securing the application, making application development more cost effective.*

Keywords: Big Data, Security Engineering, Security Requirements, Elicitation, Prioritization

1 Introduction

Security is a characteristic of software systems which ensures prevention of circumstances leading to the loss of confidentiality, integrity, and availability (CIA) of data. These three abbreviated CIA are the goals at the core of software systems security as identified by several authors [1,2,3]. The essence of security is to protect assets or resources of an organization. A database contains a company's most valuable asset, data [4,5] that requires resources to be invested in its protection. In the last few years the world has experienced a global increase in data generated from a variety of sources for instance transactions, sensor devices, websites, social networks and so forth. In a few years data has grown from hundreds of gigabytes (GB), crossing into hundreds of terabytes (TB) and into petabytes (PB) [6]. This has given birth to the term Big Data to describe these massive volumes of data and their associated management technologies. Big Data storage techniques have presented a breakthrough in achieving scalability, cost reduction, performance and flexibility in the management of Big Data. However, security has remained a daunting challenge in this technology as it was never prioritized in software development. It is imperative that profound attention be directed to exploring the Security Requirements of Big Data environments.

According to [7] databases have the highest rate of breaches among all business assets. Security challenges in databases have been discovered to be even more profound in Big Data environments. Big Data assimilate a variety of data, making them a target for attack by intruders, malicious crackers and other threats due to the criticality of data stored therein. This has led to serious security breaches leading to loss of data confidentiality, integrity and availability, a situation with adverse impact on business operations. Several researches have been carried out on Security Engineering [8,9,10]. However, none of these researches presented work in the Big Data domain.

Few researchers have considered security requirements for database related domains. Soler et. al [11] introduces a model for security requirement for data warehousing that presents a three-step process for security requirements modeling. They focus on information and quality-of-service requirements (including security) and then combine it with an approach based on Model Driven Architecture. Bertino and Sandhu [12] discuss database security, focusing mainly on various access control models. They carry out their research on relational databases, object oriented databases and XML. Similarly [13] discuss database security in terms of access control models. However, none of these papers explore Big Data security requirements. This scenario presents clear evidence that in as far as Security Engineering study is concerned, Big Data still remains unexplored yet it is one of the most vulnerable areas to security breaches in present day.

The contribution of this paper is the development of a security engineering framework for Big Data which is an adaption of the generic Security Engineering method presented [14]. The framework has various activities that start with the identification of security requirements, its analysis and prioritization with design decisions and testing. Here our

focus is on elicitation and prioritization of security requirements for Big Data. Once requirements are prioritized design decisions are taken which result in choosing optimized security mechanism. The paper is organized as in Section 2 overview of the security engineering framework is provided. Section 3 focuses on the proposal for Security Requirements Engineering in the Big Data environments and explains it with a case study of MongoDB a popular Big Data store. Finally section 4 will give the conclusions and future work.

2 Security Engineering Framework

A generic framework for Security Engineering presented in our earlier work [14] is modified for Big Data as depicted in Figure 1. It consists of phases that are **(i) Security Requirements Engineering** where security requirements as defined by Firesmith [15] are elicited along with functional requirements based on vulnerabilities inherent in functionality. Then they are prioritized and specified. **(ii) Security Design Engineering** here a Cryptographic Algorithm is identified, to mitigate the vulnerabilities which are root cause of security threats. This phase starts with the mapping of security requirements with security services, then, threats identified in previous steps are mapped to attacks. Then various environmental and communicational constraints are considered. Finally, based on attack analysis and design constraints, best suitable crypto graphical technique is selected by generating a template (iii) **Security Requirements Testing** is done to test deployed security protocol is mitigating all vulnerabilities and are protecting assets of the organization.
Here in this paper, we are concentrating only on the security requirements engineering phase, which is elaborated in the next section. Security engineering framework for big data which is shown in Figure 1 is now discussed in detail with respect to Security Requirements Engineering phase and its sub activities using the case study of MongoDB:

3.1 Security Requirements Elicitation: Using the viewpoint approach of Sommerville [16] Security Requirements are identified along with the operations. The root cause of security concerns are the vulnerabilities associated with the operations (functionalities). First various actors are identified, and then operations performed are encapsulated. Then, based on the operations, vulnerabilities inherent in the system are identified and next the security requirements are elicited. This step consists of the following activities:
(i) Generic Actor Identification: Actors are those who interact with the system such as user, administrator, and management. Actors can be of type human, cooperative (such as a DBMS) or autonomous actor (such as standalone computational software) [17]. Here, only two generic actors User (human) and Database are considered for illustration.
(ii) Operation Identification: Operations performed by actors are identified. There are four basic functions of persistent storage, which are referred by the acronym CRUD that stands for Create, Read, Update and Delete in Big Data. These are also known as database operations. Table 1 below gives an overview of the MongoDB CRUD Operations [18].

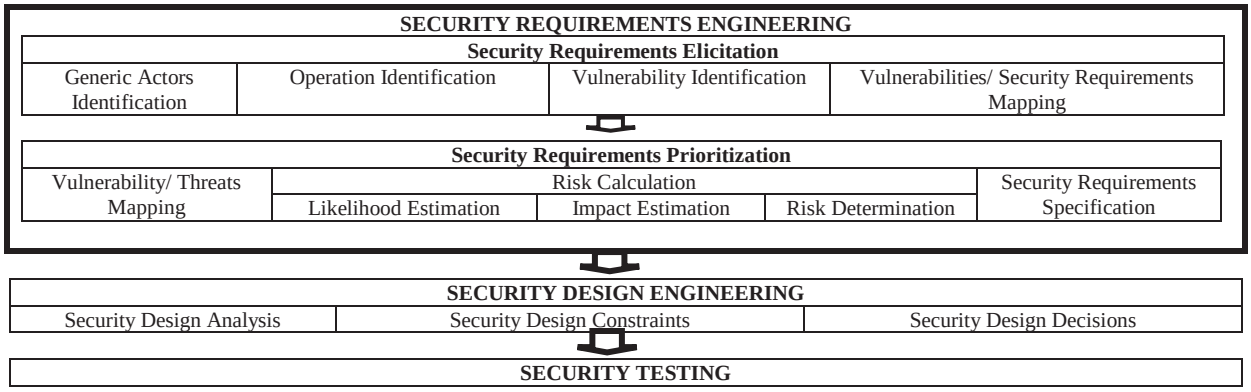


Figure 1: Framework for Security Engineering for Big Data

Table 1: MongoDB CRUD Operations

Operations	Method
Create	insert – this is the method to insert a document or documents in MongoDB updates with the upsert option- this operation accepts an "upsert" flag that modifies the behaviour of update() from updating existing documents, to inserting data.
Read	find- this is the primary method to select the documents. It also returns a cursor that contains a number of documents. findOne- this method selects a single document from a collection and returns that document, it does not return a cursor
Update	update - this method is used to modify documents in a MongoDB. By default, the update() method updates a single document, but all documents can be updated using the multi option. The update() method can either replace the existing document with the new document or update specific fields in the existing document. save- this method performs a special type of update(), depending on the _id field of the document.
Delete	the remove() method is used to delete documents from a collection.

(iii) Vulnerability identification: Vulnerability is a weakness in the system environment that a malicious attacker could exploit to cause damage to the system. It is the vulnerability that enables a threat to be exercised within the system. In a Big Data environment vulnerabilities arise due to the complexity brought on by the type and distributed nature of data involved. All vulnerabilities are depicted with a prefix V. For Big Data stores, vulnerabilities are identified by drawing and analysing the sequence diagram for each operation. With the help of sequence diagram, we get to know the exact points where the data is entered and when data is transferred or passed to another site. Sequence diagram for Create operation in MongoDB are shown in Figure 2. It is covering two methods for create operation as mentioned in [18]. But only insert part would be further considered due to space constraint. The output of this step is shown in Table 2.

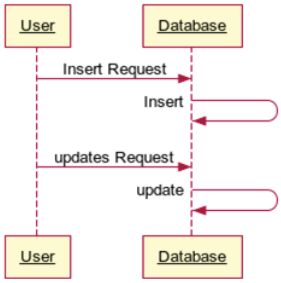


Figure 2: Sequence Diagram for Create operation in MongoDB

(iv) Vulnerabilities/ Security Requirements Mapping: After vulnerabilities are identified using a sequence diagram, security requirements are elicited to mitigate the vulnerabilities, according to a set of mapping criteria specified with a correlation matrix, shown in Table 3.

Elicited Security Requirements for CREATE operations by applying the correlation matrix are shown in Table 2. Once the security requirements elicitation process is done, we proceed to the prioritization and specification phases of the framework.

3.2 Security Requirements Prioritization and Specification:

As all requirements cannot be implemented in hand, prioritization is required. Elicited security requirements are prioritized so that depending on resources available, high priority requirements get implemented first. Risk-based method is adopted for security requirements prioritization. The steps followed in prioritization of security requirements are specified below:

(i) Vulnerabilities/Threats Mapping: Vulnerability leads to threats. Vulnerable points are breached by various potential threats. Hence mapping of vulnerability to threats is needed to be done for risk calculation and that will in turn be used to prioritize the security requirements.

Different types of threats as described in Common Criteria [19] and some additional threats are identified. Prefix T is used with threat name to make it distinguishable. A threat and vulnerability database is maintained from where the mapping of various threats to vulnerabilities is done as shown in Figure 3.

Table 2: Vulnerabilities and Security Requirements for Create Operation for User

Operations	Method	Vulnerable interaction Sequences	Vulnerabilities	Security Requirements
Create	insert()	User->Database: Insert Request	1. V.Weak_Access_Control 2. V.Unencrypted_Data 3. V.Unsecured_Network 4. V.Monitoring_Absence 5. V.Network_Partition 6. V.Breached_Firewall 7. V.Inadequate_Logging 8. Untrained_Users 9. V.Unsecured_API 10. V.Unvalidated_Input	1. Identification 2. Authentication 3. Authorization 4. Integrity 5. Privacy 6. Immunity 7. Intrusion Detection 8. Security Auditing 9. System Maintenance 10. Non-repudiation
		Database: ->Database: Insert	1. V.Unencrypted_Data 2. V.Breached_Firewall 3. V.Monitoring_Absence 4. V.Physical_Security 5. V.Misconfigurations 6. V.Unsecured_API 7. V.Physical_Security	1. Security Auditing 2. Intrusion Detection 3. Integrity 4. Privacy 5. Immunity 6. System Maintenance 7. Physical Protection
		Database ->:User: Return Confirmation	1. V.Network_Partition	1. Survivability

Table 3: Vulnerabilities and Security Requirements Correlation Matrix

Vulnerabilities \ Security Requirement	Identification Authentication Authorization	Immunity	Integrity	Intrusion Detection	Non-repudiation	Privacy	Security Auditing	Survivability	Physical Protection	System Maintenance
V.Misconfigurations							X			X
V.Weak_Access_Control	X									
V.Unencrypted_Data			X			X				
V.Breached_Firewall		X	X	X		X	X			X
V.Unsecured_API		X	X	X		X	X			
V.Unsecured_Network		X	X	X		X				
V.Network_Partition								X		
V.Unvalidated_Input		X					X			
V.Untrained_Users	X		X			X				
V.Monitoring_Absence							X			
V.Inadequate_Logging					X		X			
V.Physical_Security									X	
V.Obsolete_System										X

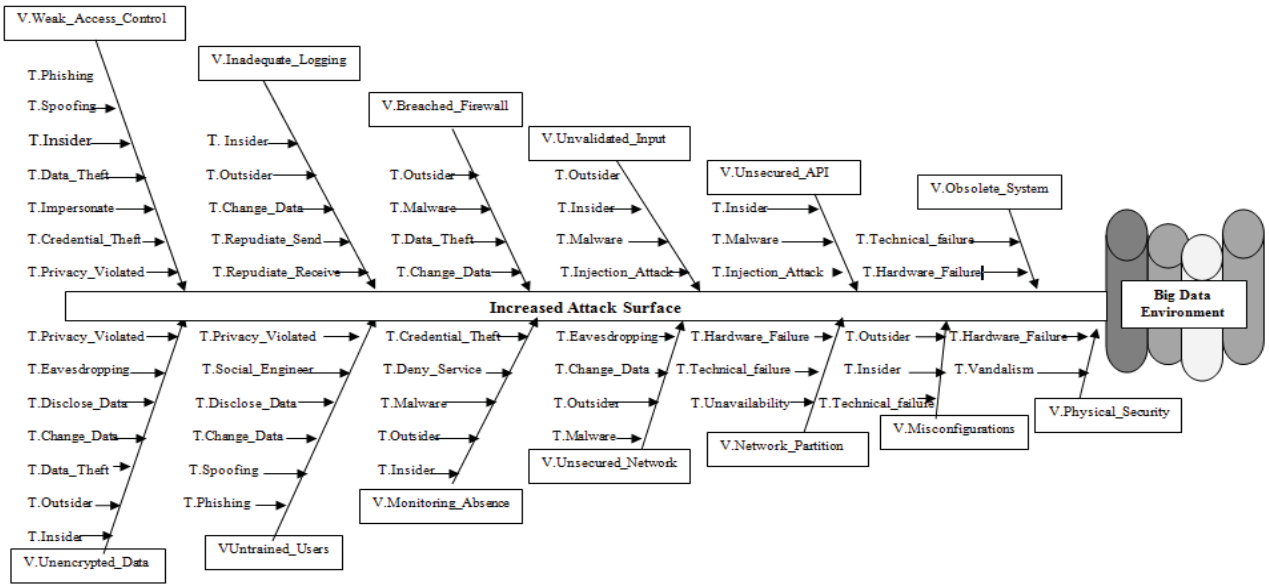


Figure 3: Vulnerabilities and Threats for Big Data Environment

(ii) Risk Calculation: The Risk can be defined in many ways. One definition says; it is the probability that a particular threat will exploit a particular vulnerability. The formula given in the OWASP [20] model for risk calculation:

*Risk = Likelihood * Impact.*

Steps involved in the risk calculation are:

(a) Likelihood Estimation: Likelihood is a measure that shows how probable a particular vulnerability is to be exploited by an attacker (threat). Likelihood ratings are estimated for vulnerabilities and threats based on the degree of satisfaction of vulnerability and threat factors. Threat factors are evaluated according to skill level, motive, opportunity and size while the likelihood is evaluated for ease of discovery, ease of exploitation, awareness and intrusion detection, each having an estimated score represented by levels Low (L), Medium (M) and High (H). Then these scores are evaluated to get one value for likelihood. A Matrix of our vulnerability/threat mapping for calculation of likelihood score is shown in Table 4.

(b) Impact Estimation: Impact refers to the consequences of a successful exploit. Impact ratings for threats and vulnerabilities are estimated. The factors considered are technical impact and business impact. Technical factors include loss of confidentiality, integrity, availability and accountability. Business impact includes financial damage, reputation damage, non-compliance and privacy violation. Next, the average impact is calculated. Same notations are used to represent the level as used by likelihood estimation and the scores as shown in Table 5.

Table 4: Likelihood Estimation

Vulnerability / Threats	V.Misconfigurations	V.Weak_Access_Control	V.Unencrypted_Data	V.Breached_Firewall	V.Unsecured_API	V.Unsecured_Network	V.Network_Partition	V.Unvalidated_Input	V.Untrained_Users	V.Monitoring_Absence	V.Inadequate_Logging	V.Physical_Security	V.Obsolete_System
T.Change_Data		H	H	M		M			H		L		
T.Data_Theft		H	H	M									
T.Deny_Service										M			
T.Disclose_Data			H						L				
T.Impersonate		H											
T.Injection_Attack					L			L					
T.Fraud		M									H		
T.Privacy_Violated		H	H						L				
T.Eavesdropping			H			M							
T.Credential_Theft		H								M			
T.Social_Engineer									L				
T.Phishing		L							L	L			
T.Spoofing		L							L	L			
T.Repudiate_Receive											L		
T.Repudiate_Send											L		
T.Insider	L	H	M		M			L		M	H	M	
T.Outsider	M		H	M		H		M		M	L	L	
T.Technical_failure	L						L						L
T.Hardware_Failure							M					L	L
T.Vandalism												L	
T.Malware				M	M	M		M		H			
T.Unavailability							H						

Table 5: Impact Estimation

Vulnerability / Threats	V.Misconfigurations	V.Weak_Access_Control	V.Unencrypted_Data	V.Breached_Firewall	V.Unsecured_API	V.Unsecured_Network	V.Network_Partition	V.Unvalidated_Input	V.Untrained_Users	V.Monitoring_Absence	V.Inadequate_Logging	V.Physical_Security	V.Obsolete_System
T.Change_Data		H	H	H		H			H		H		
T.Data_sTheft		H	H	H									
T.Deny_Service										H			
T.Disclose_Data			H						H				
T.Impersonate		H											
T.Injection_Attack					H			H					
T.Fraud		H									H		
T.Privacy_Violated		H	H						H				
T.Eavesdropping			H			H							
T.Credential_Theft		H								H			
T.Social_Engineer									H				
T.Phishing		H							H	H			
T.Spoofing		H							H	H			
T.Repudiate_Receive											M		
T.Repudiate_Send											L		
T.Insider	H	H	H		H			H		H	H	H	
T.Outsider	H		H	H		H		H		H	H	H	
T.Technical_failure	H						H						M
T.Hardware_Failure							M					M	L
T.Vandalism												M	
T.Malware				M	M	M		M		M			
T.Unavailability							H						

(6) Risk Determination: After calculation of scores for both likelihood and impact. The Severity given in Table 6 is used to determine risk value.

Now security requirements are prioritized based on the risk rating, results are shown in Table 7 below.

Table 6: Overall Risk Severity

Overall Risk Severity				
Impact	HIGH	Medium	High	High
	MEDIUM	Low	Medium	High
	LOW	Low	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Table 7: MongoDB security requirements prioritization

Operations	Method	Vulnerable interaction Sequences	Security Requirements	Vulnerabilities	Priority
Create	insert()	User->Database: Insert Request	Identification Authentication Authorization	V.Weak_Access_Control V.Untrained_Users	High
			Integrity	V.Weak_Access_Control V.Unencrypted_Data V.Breached_Firewall V.Unsecured_Network V.Untrained_Users V.Inadequate_Logging	High
			Privacy	V.Unencrypted_Data V.Breached_Firewall V.Untrained_Users V.Unsecured_Network V.Unsecured_API	High
			Immunity	V.Unsecured_API V.Unvalidated_Input V.Unsecured_Network V.Breached_Firewall	High
			Intrusion Detection	V.Breached_Firewall V.Monitoring_Absence V.Unsecured_Network V.Unsecured_API	High
			Security Auditing	V.Monitoring_Absence V.Inadequate_Logging V.Misconfigurations V.Breached_Firewall V.Unsecured_API V.Unvalidated_Input	High
			System Maintenance	V.Obsolete_System V.Misconfigurations V.Breached_Firewall	Medium
			Non-repudiation	V.Inadequate_Logging	Medium
		Database: ->Database: Insert	Security Auditing	V.Breached_Firewall V.Monitoring_Absence V.Inadequate_Logging V.Misconfigurations V.Unsecured_API	High
			Intrusion Detection	V.Breached_Firewall V.Monitoring_Absence V.Unsecured_API	High
			Integrity	V.Unencrypted_Data V.Breached_Firewall	High
			Privacy	V.Unencrypted_Data V.Breached_Firewall V.Unsecured_API	High
			Immunity	V.Unsecured_API V.Unvalidated_Input V.Unsecured_Network V.Breached_Firewall	High
			System Maintenance	V.Obsolete_System V.Misconfigurations V.Breached_Firewall	Medium
			Physical Protection	V.Physical_Security	Medium
		Database ->:User: Return Confirmation	Survivability	V.Network_Partition	High

3.3 Security Requirements Specification: The Security Requirements Specification is the final phase of our framework. The security requirements specification document would have all information related to security requirements engineering activity such as list of actors identified, operation and its description, list of prioritized security requirements and so on.

3 Conclusions and Future Work

Security Requirements Engineering is of paramount importance in software engineering and in building Big Data systems that are robust and able to withstand various forms of threats and attacks. Securing Big Data environments is not an easy task due to the complexity of these environments. However, careful consideration of Security Requirements in the development of Big Data store can ensures that data is protected at the source. It will be worthwhile to have adequate security built into the databases as they are developed which ultimately lowers costs.

We have illustrated with the help of a case study how security requirements of Big Data can be elicited, analyzed and prioritized. This is a generic approach which can be used for design of Big Data. We have analyzed MongoDB and identified various security issues, which could be eliminated by our approach. Various vulnerabilities identified, such as V.Weak_Access_Control, V.Unencrypted_Data are eliminated by implementation of Identification, Authentication and Authorization security requirements. The approach can be used for the design of new Big Data. A tool has been prepared for the approach.

Based on the security requirements and constraints (environmental, design) a cryptographic algorithm will be identified and then testing is performed to validate the system security concerns. As a part of future work, more case studies of the Big Data stores will be explored to verify whether our framework can be applied to any Big Data. Furthermore, work on other phases of Security Engineering is under processing.

4 References

[1] Amoroso, E. G., Fundamentals of Computer Security Technology, Prentice-Hall, ISBN: 0-13-305541-8, 1994

[2] T M Kiran Kumar, "A Road Map to the Software Engineering Security", IEEE 2009.

[3] Toktam Ramezani Farkhani, Mohammad Reza Razzazi Examination and Classification of Security Requirements of Software Systems", 2006 IEEE

[4] Diya Soubra, "The 3Vs that define Big Data", Posted on July 5, 2012 on Data Science Central, http://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data, Accessed on 05/04/2014

[5] 3Vs (volume, variety and velocity), Posted by Margaret Rouse, http://whatis.techtarget.com/definition/3Vs, Accessed on 05/04/2014

[6] Joseph McKendrick, Research Analyst, "The Petabyte Challenge: 2011 IOUG Database Growth Survey", Produced by Unisphere Research, A Division of Information Today, Inc. August 2011

[7] Avita Katal, Mohammad Wazid, R H Goudar, "Big Data: Issues, Challenges, Tools and Good Practices", IEEE 2013

[8] Charles B. Haley, Robin Laney, Jonathan D. Moffett, "Security Requirements Engineering: A Framework for Representation and Analysis", IEEE Transactions On Software Engineering, Vol. 34, No. 1,), pp.133-153. January/February 2008

[9] N. R. Mead, T. Stehney, "Security Quality Requirements Engineering (SQUARE) Methodology," Proc. of the 2005 workshop on software engineering for secure systems-building trustworthy applications, Missouri, USA, 2005, pp.1-7.

[10] Daniel Mellado, Eduardo Fernández-Medina, Mario Piattini, "Security Requirements EngineeringProcess for Software Product Lines: A Case Study", The Third International Conference on Software Engineering Advances IEEE 2008

[11] Emilio Soler, Veronika Stefanov, Jose-Norberto Maz´on, Juan Trujillo, Eduardo Fern´andez-Medina, Mario Piattini, "Towards Comprehensive Requirement Analysis for Data Warehouses: Considering Security Requirements", The Third International Conference on Availability, Reliability and Security, IEEE 2008

[12] Elisa Bertino, and Ravi Sandhu,, "Database Security—Concepts, Approaches, and Challenges", Transactions On Dependable And Secure Computing, Vol. 2, No. 1, IEEE 2005

[13] Leon Pan, "A Unified Network Security and Fine-Grained Database Access Control Model", Second International Symposium on Electronic Commerce and Security, IEEE 2009

[14] Gupta, D., Chatterjee, K., De, A.: A Framework for Development of Secure Software. CSI Transaction on ICT (2013)

[15] Donald Firesmith: Engineering Security Requirements, in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pages 53-68.

[16] Sommerville, I.: Software Engineering. Pearson Education, London (2003) ISBN-8129708671.

[17] Michael S. Ware John B. Bowles Caroline M. Eastman, "Using the Common Criteria to Elicit Security Requirements with Use Cases"

[18] MongoDB CRUD Operations Introduction Release 2.2.7, April 15, 2014.

[19] "Common Criteria for Information Technology Security Evaluation", version 3.1, reversion 1, Sep. 2006.

[20] OWASP Risk Rating Methodology, http://www.owasp.org/index.php/OWASP_risk_rating_methodology , Accessed on 30 May 2014.