Stefan Biffl
Dietmar Winkler
Johannes Bergsmann (Eds.)

LNBIP 94

# Software Quality

## Process Automation in Software Development

4th International Conference, SWQD 2012
Vienna, Austria, January 2012
Proceedings

2012 software quality days

Springer

# Lecture Notes in Business Information Processing 94

Stefan Biffl
Dietmar Winkler
Johannes Bergsmann (Eds.)

# Software Quality

## Process Automation in Software Development

4th International Conference, SWQD 2012
Vienna, Austria, January 17-19, 2012
Proceedings

Springer

Volume Editors

Stefan Biffl
Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstrasse 9/188
1040 Vienna, Austria
E-mail: stefan.biffl@tuwien.ac.at

Dietmar Winkler
Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstrasse 9/188
1040 Vienna, Austria,
E-mail: dietmar.winkler@tuwien.ac.at

Johannes Bergsmann
Software Quality Lab GmbH
Gewerbepark Urfahr 30, BG4
4041 Linz, Austria
E-mail: johannes.bergsmann@software-quality-lab.at

ISSN 1865-1348 e-ISSN 1865-1356
ISBN 978-3-642-27212-7 e-ISBN 978-3-642-27213-4
DOI 10.1007/978-3-642-27213-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011943099

ACM Computing Classification (1998): D.2, K.6

*Typesetting:* Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# Message from the General Chair

The Software Quality Days (SWQD) conference and tools fair started in 2009 and has grown into one of the biggest conferences on software quality in Europe within a strong community. The program of the SWQD conference is designed to encompass a stimulating mixture of practical presentations and new research topics in scientific presentations as well as tutorials and an exhibition area for tool vendors and other organizations in the area of software quality.

This professional symposium and conference offers a range of comprehensive and valuable opportunities for advanced professional training, new ideas, and networking with a series of keynote speeches, professional lectures, exhibits, and tutorials.

The SWQD conference is suitable for anyone with an interest in software quality, such as test managers, software testers, software process and quality managers, product managers, project managers, software architects, software designers, user interface designers, software developers, IT managers, development managers, application managers, and others with similar roles.

January 2012 Johannes Bergsmann

# Message from the Scientific Program Chair

The 4th Software Quality Days (SWQD) conference and tools fair brought together researchers and practitioners from business, industry, and academia working on quality assurance and quality management for software engineering and information technology. The SWQD conference is one of the largest software quality conferences in Europe.

Over the past few years a growing number of scientific contributions have been submitted to the SWQD symposium. This was the first year with a dedicated scientific track published in scientific proceedings. In total we received from researchers across Europe 18 high-quality submissions, which were peer-reviewed by three or more reviewers. Out of these submissions, the editors selected seven contributions as full papers, with an acceptance rate of 39%. Authors of the best papers will be invited to submit extended versions of their papers to a special section in the *Software Quality* journal. Further, six short papers, which represent promising research directions, were accepted to spark discussions between researchers and practitioners at the conference.

Main topics from academia and industry were focused on systems and software quality management methods, improvements of software development methods and processes, the latest trends in software quality, and testing and software quality assurance.

This book is structured according to the sessions of the scientific track following the guiding conference topic *"Process Automation in Software Development"*:

- Software Product Quality
- Software Engineering Processes
- Software Process Improvement
- Component-Based Architectures
- Risk Management
- Quality Assurance and Collaboration

January 2012 Stefan Biffl

# Organization

SWQD 2012 was organized by Software Quality Lab GmbH and the Vienna University of Technology, Institute of Software Technology and Interactive Systems.

## Organizing Committee

### General Chair

| | |
|---|---|
| Johannes Bergsmann | Software Quality Labs GmbH |

### Scientific Chair

| | |
|---|---|
| Stefan Biffl | Vienna University of Technology |

### Proceedings Chair

| | |
|---|---|
| Dietmar Winkler | Vienna University of Technology |

### Organizing and Publicity Chair

| | |
|---|---|
| Petra Bergsmann | Software Quality Labs GmbH |

## Program Committee

SWQD 2012 established an international committee of well-known experts in software quality and process improvement to peer-review the scientific submissions.

| | |
|---|---|
| Muhammad Ali Babar | IT University of Copenhagen, Denmark |
| Maria Teresa Baldassarre | University of Bari, Italy |
| Antonia Bertolino | ISTI CNR, Italy |
| Miklos Biro | Dennis Gabor College Budapest, Hungary |
| Ruth Breu | University of Innsbruck, Austria |
| Michel Chaudron | Leiden University, The Netherlands |
| Deepak Dhungana | Siemens Corporate Technology Research, Austria |
| Schahram Dustdar | TU Vienna, Austria |
| Frank Elberzhager | Fraunhofer IESE, Kaiserslautern, Germany |
| Gordon Fraser | University of Saarland, Germany |
| Christian Frühwirth | Aalto University, Finland |
| Marcela Genero | University of Castilla-La Mancha, Spain |
| Tony Gorschek | Blekinge Institute of Technology, Sweden |
| Volker Gruhn | University of Duisburg-Essen, Germany |

## Sub-Reviewers

# Table of Contents

# Quality Assurance in Model-Based Software Development - Challenges and Opportunities -

Michel R.V. Chaudron

Leiden University
Leiden Institute of Advanced Computer Science, The Netherlands
chaudron@liacs.nl

**Abstract.** Modeling is a common practice in modern day software engineering. Since the mid 1990's the Unified Modeling Language (UML) has become the de facto standard for modeling software systems. The UML is used in all phases of software development – ranging from the requirements phase to the maintenance phase. However, the manner in which the UML is used differs widely from project to project and from developer to developer. This illustrates an apparent lack of quality awareness in the use of UML. In this paper I will discuss the challenges and opportunities there are for using quality assurance for software modeling for improving the quality and productivity of software development.

## 1 The State of the Practice and the Need for Quality Assurance in Model Based Software Development

Over the past 15 years I have visited many software development projects that use some form of modeling. I have collected both quantitative as well as qualitative data about the way in which those projects performed software modeling and about the impact of their modeling practice on overall project productivity and quality. Some of the key findings from these studies are the following (these are elaborated in [1]):

- There is a large variation in styles of software modeling between different developers and across different projects – also within individual organizations.
- In virtually all projects that we visited:
    - the UML models contained a large amount of incompleteness and inconsistency
    - no processes or techniques were applied for quality assurance of UML models

This state of the practice suggests that there are significant opportunities for achieving improvements for quality assurance for the modeling activities in software projects.

In the remainder of this paper, I will discuss:

- Economics of modeling: Does modeling actually help in creating better software? How much modeling is enough?

- What are practical ways of starting with quality assurance in model-based software development?
- Promises from the state of the art in software model quality assurance techniques and challenges for the future

## 2 Does Investing in Modeling and Quality of Modeling Make Business Sense?

In his seminal book Software Engineering Economics Barry Boehm reports on the relation between the stage of a project (starting, middle, finishing) and the cost of repairing defects. The relation is such that the cost of repairing increases exponentially with the progress of a project. More precisely this principle is nowadays formulated as that the cost of repairing a defect grows exponentially with the life-time of a defect.

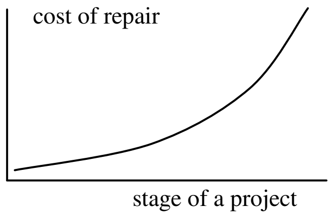

**Fig. 1.** Relation between stage of a project and cost of repair

When we look at the activities of a software development project that have the largest impact on the final quality on the system, then these are the requirements and architecting/design activities. Hence, from first principles it makes economic sense to ensure quality of the early products of software projects.

Next, one can ask: does it help to model our design in a systematic manner using UML? I will draw from some empirical studies to suggest that using models indeed makes business sense:

In a controlled experiment subjects were shown UML models that either lacked some information or contained some inconsistencies versus models that were complete. It was found that for the incomplete models, subjects had much larger variation in the interpretation of the meaning of the model [2]. Clearly, this variation in interpretation increases the chances of miscommunication and misinterpretation.

Subsequently we tried to find evidence in actual industrial software development projects. To this end, we studied the defects that we found in the project repository of a medium size industrial project [3]. We counted the number of defects per class (in the implementation), and subsequently we looked at the manner in which each of these classes was modeled in the UML design. This showed that classes for which more detail was present in the UML model (to be precise in the sequence diagrams) contained fewer defects than classes for which less detail was present in the UML model. At the same time, this study also showed that there is an effect of diminishing returns in model quality (See Fig 2): at some point, adding more detail to a model no longer correlates with a significant decrease in the expected number of defects in that class. This study shows that the quality of sequence diagrams is important when UML is used as blueprint for implementation.

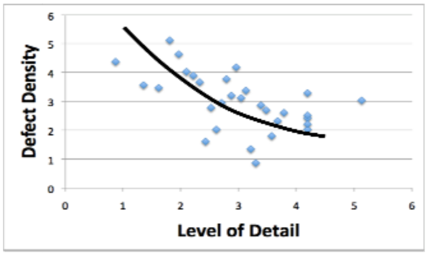

**Fig. 2.** Relation between Detail in Sequence Diagram and Defect Density

Complementary to the support for modeling implied by these quantitative studies, there are also more subjective pieces of evidence – obtained through surveys and interviews – in which designers and programmers state that they feel that the use of modeling aids in achieving a common understanding of a design within an team and that modeling helps as an aid in communication. These benefits ultimately lead to a positive effect on productivity and quality [4].

Additional evidence that supports the importance of good documentation comes from an interesting corner: the documentation-averse agile software development community: a recent study amongst agile development teams showed that they would prefer to have more documentation than they normally produce [5].

## 3 Practical Lightweight Methods for Starting Quality Assurance for Model Based Software Development

In this section I will discuss some lightweight, easy to use methods for quality assurance of UML models.

### 3.1 Quality of UML Models

The basis for any quality assessment is a quality model. In the area of software, several quality models exist. All of them define quality through a hierarchical composition which has in the leave-nodes some measurable characteristic of the software. This same approach can be applied for establishing a quality model for UML models. Rather than measuring properties in the source code, properties must be measured from the UML models. Some types of measurements (a.k.a. metrics) carry over fairly straightforwardly from code to models. However, some other metrics may need to be adapted. Furthermore, UML models also offer an opportunity for defining new metrics that cannot be determined from the source code. Consider for example a measure for the complexity or criticality of a use case by counting the number of sequence diagrams that support a use case.

Quality models for software used to take a "one-size-fits-all" approach. In a recent paper, we propose that the quality model for UML models should be tailored to the purpose of the task that the software-model has to support [6]. In particular, a team should determine how the models are aimed to be used or more pragmatically for which tasks they are actually most used. Figure 3 shows an example of a quality model for UML that considers maintenance and development as key activities.

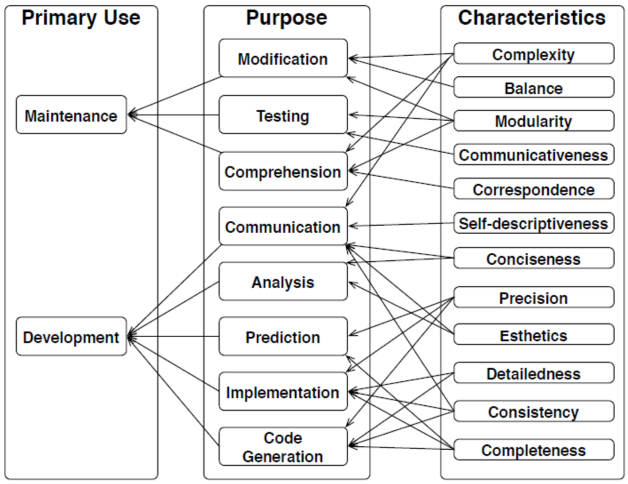

**Fig. 3.** Example Quality Model for Software Models

## 3.2 Assessing Correspondence between Source Code and UML Model

A key factor for harvesting the benefits of modeling are to ensure that the implementation actually conforms (to a large degree) to the UML design. The degree in which an implementation follows a design depends amongst other on the degree of detail and completeness in which the design was modeled. In practice we see that UML designs most often contain between 20% and 50% of the classes of the ultimate implementation. When asked how designer chose what to include in their models, designers state that they focus on complex and critical parts [7].

In line with the earlier finding that there are diminishing returns for raising the quality level of a model after some point, we should expect that there implementation may deviate somewhat from the model. This may be because the model describes the system only at a high level of abstraction and hence omits some details, or because designers in the implementation phase decide on alternative implementations (which may or may not be improvements over the design). For monitoring the compliance of the implementation to the design, we need a method that does not aim to find a perfect match between implementation and design, but is robust against differences and tries to provide insight in the severity of potential deviations.

Figure 4 shows a graph generated by Van Opzeeland [8] who provides one possible approach for a robust comparison between UML model and implementation.

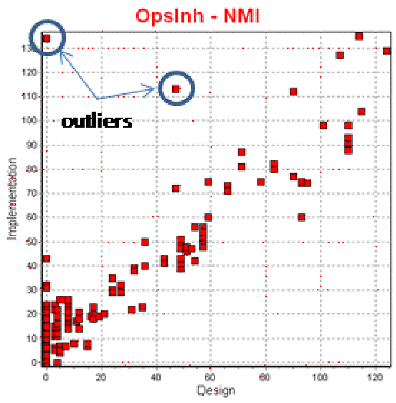

**Fig. 4.** Graph showing correspondence between Design and Implementation

The method works as follows: first a mapping is created between classes in the implementation to classes in the UML model (for non-OO languages, this would also work for functions/procedures or components). This step is greatly facilitated if the same naming conventions are used in design and implementation. Subsequently, a set of metrics (e.g. coupling, size (number of methods, number of attributes), depth-of-inheritance) is computed for implementation classes and corresponding UML-classes. Then, for each class one point in draws in the graph where the *x*-coordinate is the metric for that class in the design and the *y*-coordinate is the metrics for the corresponding class in the implementation. The resulting graph shows a line *x*=*y* when there is a perfect mapping between design and implementation. In practice we see a pattern that shows a linear relationship within some bandwidth. This pattern enables the easy visual identification of outliers which form the largest deviation between design and implementation and thus are the most important candidates for scrutiny.

## 3.3 Low Hanging Fruit for Quality Assurance for Model Based Software Development

It has become common for a set of quality assurance methods to be part of the normal operating procedure of many software projects: version control, coding and layout conventions. However, while the tooling infrastructure for such techniques is already in place, these common techniques are not also applied to the modeling activities and/or artifacts. Hence, some low-hanging fruit for a project that used models in their software development are to use the following:

- **Use coding, naming and layout conventions for models:** If you are going to use naming conventions in the coding activities anyway, why not already start using these conventions in the modeling and design activities? Using such conventions will help achieve:

  - more uniform look and feel of the models, which increases understandability and identification of anomalies in the design.
  - easier traceability between code and design. This encourages closer following of the design principles in the implementation. Also it simplifies assessing the correspondence/differences between the design and implementation (as described in section 3.2).

  A good starting point for defining such conventions is the set of guidelines on 'UML Style' by Fowler [9]. An additional consideration for modeling conventions is to standardize on which design patterns to use and how to represent these in the UML model.

  On the topic of traceability: different tools provide different support for traceability between use cases and sequence diagrams. In practice, if a tool does not support such traceability, then it is quite easy to arrange this by systematic organization of diagrams in directories: e.g. sequence diagrams can be stored in subdirectories of the associated use cases. If the traceability between classes (via sequence diagrams) to use cases can be established, this can have advantages for monitoring compliance of the implementation to the requirements, and thus can help in acceptance testing.

- **Use Version Control for models:** there is admittedly a challenge for version control systems (VCS) to cater for model-based development, as most VCS's are code-oriented. However, even a crude approach where subsequent versions of a model are checked in as a whole already add value to a project (e.g. for tracing progress, but also as a means of disseminating design knowledge). Challenges for VCS's are to better cater for models as being the unit of version control which have different properties than source code. Some ways in which models differ from source code are, amongst others: layout/formatting:

  - the orientation of class-boxes and lines may have a meaning and thus needs to be stored as part of the model;
  - modularity: there are no established way of modularizing large models while maintaining all relations between different parts;
  - variability: if a system-design comes with variants (through variability points), e.g. as part of a product family, dedicated approaches are needed for storing all related information of variants and possible compatible versions of different model of features.
  - differencing and merging: while the *diff* program works well for source code, specialized algorithms need to be developed for supporting the analyses of differences between different versions of a model;
  - traceability: ideally versions of models are linked to corresponding, but potentially varying parts of the implementation source code.

However, for most practical purposes pragmatic solutions exist.

- **Use Reviews and Inspections:** For source code, reviews and inspections have been shown to be very cost effective: they require low effort and deliver high returns. Also, this practice is quite scalable in effort: A quick and dirty way of doing reviews is to just ask a colleague to read the design and report issues. A more advanced implementation of the inspection practice uses an established set of guidelines or a checklist to go over a design systematically. The mere fact that you plan for a review and inspection is a good way of ensuring that it takes place, rather than getting skipped under the inevitable schedule-pressure of a project.

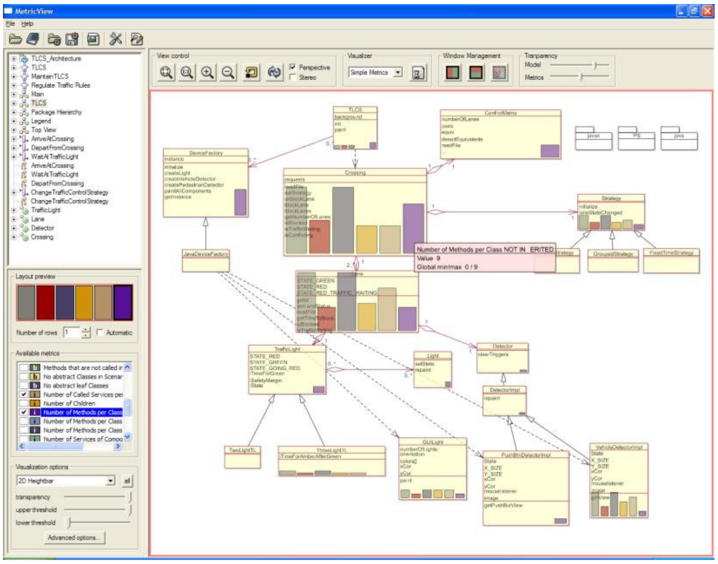

**Fig. 5.** Screenshot of the MetricView tool

- **Use Software Metrics for Design:** Software metrics are widely used for source code. The premise of software metrics is that quantitative measurements about the software are an indicator of quality properties. For source code, many metrics are around. An often used metrics in quality assurance is cyclomatic complexity, for which high values are an indicator of poor modularization, low understandability and high defect-proneness. Analogously, metrics can be applied in the (UML) design stage. For example, a high value for a coupling metrics (which counts the degree of interdependencies of a class/component with other classes), may be an indicator for an unbalanced design, too much logic lumped together, which typically leads to high maintenance effort. Some UML modeling tools provide basic support for computing design metrics. However, there are very affordable tools around such as SDMetrics[1] which are dedicated and thus specialized in computing metrics for UML models. Such dedicated metric tools

[1] www.sdmetrics.com

provide a much richer set of metrics and can be easily tuned to project preferences. In our own research we have developed a tool called MetricView [10] that visualizes design metrics by projecting then on top of the UML diagrams (see Figure 5). This provides intuitive feedback on the design.

UML metrics can be integrated (via a version management system) in a software development dashboard (like Sonar) that also manages code metrics. The aforementioned correspondence checking method shows how using the combination of source code metric and UML model metrics can be beneficial.

## 4 What Does the Future Have in Store for Quality Assurance in Model Based Software Development

A first challenge is in the area of model-based testing: economically it makes sense to reuse a model that has been used to design a system also in the testing phase, e.g. for generating tests from. However, current model-based testing approaches require a high level of detail and completeness of the model as prerequisite before test can be generated. A challenge for the modeling and testing researchers is to turn into use for the testing activities the models that were produced in the design phase that are typically not complete nor highly detailed.

Another approach through with the modeling effort from the design stage can be capitalized is through the analysis of the extra-functional properties of a system. Promising progress is made in approaches where a regular UML model is decorated by annotations that provide information about the load profile and resource claims of the operations of a system. Subsequently, based on this enriched model, performance of the system can be analyzed. The research in this direction is aiming to enable the concurrent analysis of multiple extra-functional properties (including performance, safety, reliability, cost, security) based on a single core model which is decorated with different adornments.

A third challenge is to rank the severity of defects based on understanding the design of a system. Also, here UML models which include a traceability linking between classes via sequence diagrams to use cases may provide to be a richer source of information about a system's operating context and thus for inferring information about severity of defects in the design.

## 5 Conclusion

Modeling is a common practice in software development and will become more common with the adoption of model-driven technology. The industrial practice of quality assurance for software modeling in general and UML in particular is far behind what is practically possible. Currently practical methods, tools and guidelines are available for quality assurance of UML models. These methods promise to improve the quality and productivity of projects at low cost.

## References

1. Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: In Practice: UML Software Architecture and Design Descriptions. IEEE Software 23(2), 40–46 (2006)
2. Lange, C.F.J., Chaudron, M.R.V.: Effects of defects in UML models: an experimental investigation. In: Proc. of the 28th Int. Conference on Software Engineering (ICSE 2006), Shanghai, pp. 401–411. ACM (2006)
3. Nugroho, A., Flaton, B., Chaudron, M.R.V.: Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density. In: Busch, C., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 600–614. Springer, Heidelberg (2008)
4. Nugroho, A., Chaudron, M.R.V.: A survey into the rigour of UML use and its perceived impact on quality and productivity. In: Empirical Software Engineering and Measurement (ESEM 2008), pp. 90–99. ACM (2008)
5. Stettina, C.J., Heijstek, W.: Necessary and Neglected? An Empirical Study of Internal Documentation in Agile Software Development Team. In: 29th ACM International Conference on Design of Communication. ACM (2011)
6. Lange, C.F.J., Chaudron, M.R.V.: Managing Model Quality in UML-based Software Development. In: Workshop on Softwre Technology and Engineering Practice, pp. 7–16. IEEE Computer Society, Washington (2005)
7. Nugroho, A., Chaudron, M.R.V.: A survey of the practice of design-code correspondence amongst professional software engineers. In: Empirical Software Engineering & Measurement, pp. 467–469. IEEE (2007)
8. van Opzeeland, D.J.A., Lange, C.F.J., Chaudron, M.R.V.: Quantitative Techniques for the Assessment of Correspondence between UML Designs and Implementations. In: Proc. 9th QAOOSE Workshop, ECOOP 2005 (2005)
9. Ambler, S.W.: The Elements of UML2.0 Style. Cambridge University Press (2005)
10. Lange, C.F.J., Chaudron, M.R.V., Wijns, M.A.J.: Supporting Task-Oriented Modeling using Interactive UML Views. Journal of Visual Languages and Computing 4, 399–419 (2007)

# Quality Driven Software Architecture

Peter Hruschka

Atlantic Systems Guild, Langenbruchweg 71
52080 Aachen, Germany
peter@systemsguild.com

**Abstract.** This short paper introduces "quality driven software architecture" (QDSA) as a method to ensure qualities such as maintainability, modularity, scalability, or extensibility in software architectures and emphasizes the need for a person in charge (i.e. the software architect) to actively manage and control such qualities.

**Keywords:** Software Architecture, Quality Goals, Architecture Development Process, Evaluating Software Architecture.

## 1 Introduction

Users normally expect a certain qualities from their software systems, even if they do not explicitly ask for it. They often take issues like maintainability, modularity, scalability, extensibility, etc. for granted.

There are various ways to come up with key design decisions for a software architecture. A popular approach is work "domain driven" (as e.g. elaborated in [1] or [2]), to start with fundamental architectural patterns (e.g. [3]) or answer six key questions to create a first sketch of the architecture [4]. While all these approaches have their merits, they often ignore quality requirements too long in the process.

Therefore, we suggest that software architects should start from explicitly negotiated quality goals (or architecture goals). Those are often a subset of non-functional requirements. Despite all the attention that requirements engineering got in industry over the last 15 years the non-functional requirements still tend to be ignored, considered obvious or treated as orphans. And parts of them, the so-called internal qualities (like extensibility, scalability, modifiability, reusability, …) are sometimes in conflict with the project goals and constraints (like time to market, costs, …). Therefore, QDSA makes it the software architect the advocate for such longer term architectural goals. Thus, he or she is an excellent sparring partner for the project manager who is striving to achieve the project goals. Project goals are to determine whether the project was successful, while architecture goals are to determine whether the solutions has been structured in a way to achieve those longer term goals. Successful product development should ensure that both sets of goals are met and – if there is a conflict between them – that this conflict is openly discussed and resolved early on.

## 2 Quality Tree as a Starting Point

Clements et. al. [5] describe an excellent process (ATAM – Architecture Tradeoff Analysis Method) to determine how well a software architecture meets given quality goals. The key is to structure the quality goals in form of a quality tree. Concrete scenarios form the leaves of the tree, prioritized from two different points of view, the business value and the architectural challenge (cf. example in Figure 1). Those with high marks in both areas are used by experts to discuss design decisions and suggest measures for the "weak spots".

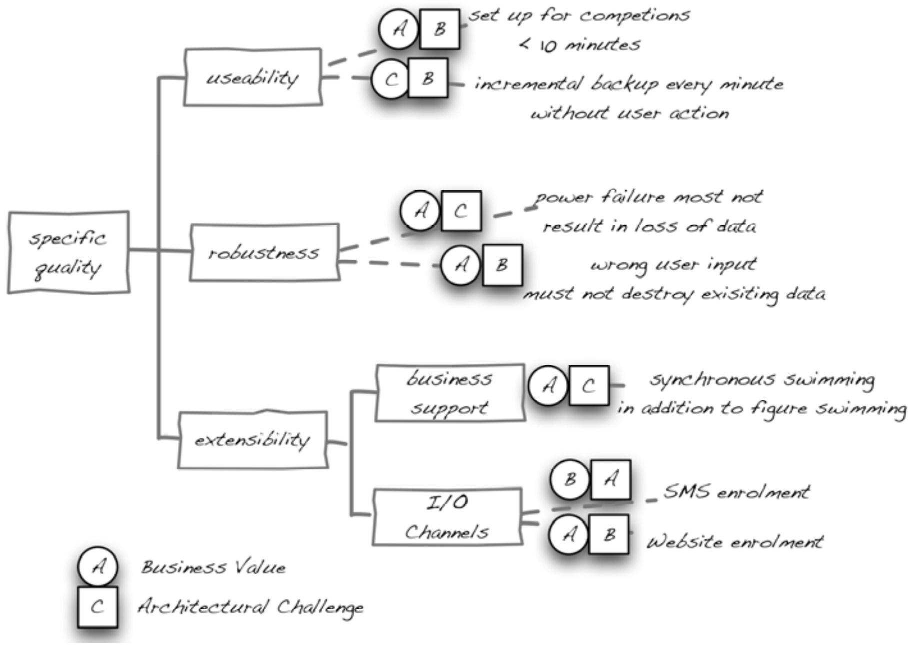
specific quality
useability
robustness
extensibility
business support
I/O Channels
A B set up for competions < 10 minutes
C B incremental backup every minute without user action
A C power failure most not result in loss of data
A B wrong user input must not destroy exisiting data
A C synchronous swimming in addition to figure swimming
B A SMS enrolment
A B Website enrolment
A Business Value
C Architectural Challenge

**Fig. 1.** Example for a Quality Tree with Scenarios

## 3 Changing ATAM to a Constructive Process

In the QDSA-approach of ARC42 [6], we suggest to use the ATAM techniques as part of the architect's normal, iterative architecture tasks (cf. figure 2).

The task "clarify requirements & constraints" – among other things – is to create the quality tree. This encourages the software architect to gain an excellent understanding of the requirements, especially the non-functional ones, which are often neglected in requirements documents. Since these quality requirements are the essential design drivers a solid understanding prior to making key architecture decisions is very helpful. In the two key tasks in the middle of figure 2 the software architect uses this detailed knowledge about the design drivers to come up with an adequately balanced design, always aware of the potential tradeoffs he or she has to make.

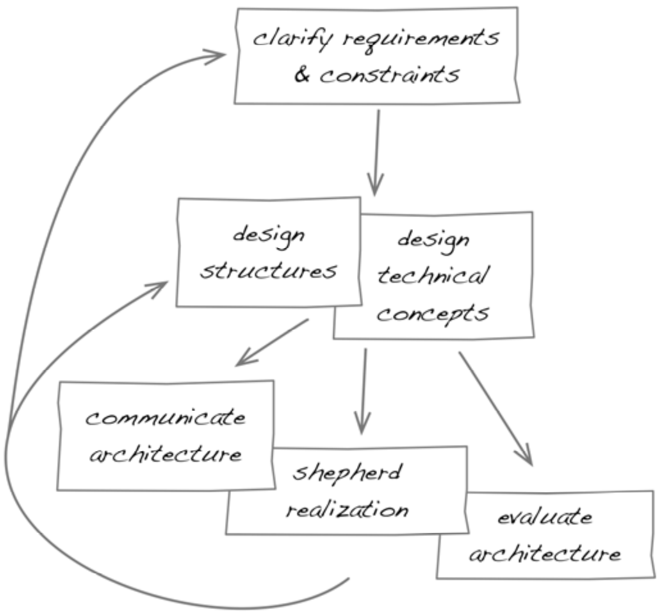

**Fig. 2.** The Iterative Architecture Development Cycle of ARC42

The three feedback tasks at the bottom of figure 2, especially the task “evaluate architecture” constantly check alignment with the defined goals. While ATAM is more of a one-shot task, these task becomes part of everyday work. Normally an hour every two weeks for evaluating is enough to stay on track.

## 4 Summary and the Way Ahead

The next steps in the development of QDSA are to support software architects further by suggesting best practices, i.e. constructive strategies and policies for selected quality goals to be applied under given constraints in the tasks “design structures” an “design technical concepts”. Then the task “evaluate architecture” should become a formal routine quality check, since it should not come up with any surprises.

**Acknowledgments.** Thanks to Gernot Starke for jointly developing the QDSA approach and the ARC42 portal for architects.

## References

1. Evans, E.: Domain Driven Design. Addison Wesley (2003)
2. Nilsson, J.: Applying Domain Driven Design & Patterns. Addison Wesley (2006)
3. Buschmann, F., Henney, K., Schmidt, D.: Pattern Oriented Software Architecture. A Pattern Language for Distributed Computing, vol. 4 (2007)
4. Starke, G., Hruschka, P.: Software-Architektur kompakt, 2nd edn. Springer, Heidelberg (2011) (in German)
5. Clements, P., et al.: Evaluating Software Architectures. Addison Wesley (2001)
6. The ARC42 Portal for Software Architects, `http://www.arc42.com`

# On the Benefit of Automated Static Analysis for Small and Medium-Sized Software Enterprises

Mario Gleirscher[1], Dmitriy Golubitskiy[1], Maximilian Irlbeck[1], and Stefan Wagner[2]

[1] Institut für Informatik, Technische Universität München, Germany
{gleirsch,golubits,irlbeck}@in.tum.de
[2] Software Engineering Group, Institute of Software Technology, University of Stuttgart, Germany
stefan.wagner@informatik.uni-stuttgart.de

**Abstract.** Today's small and medium-sized enterprises (SMEs) in the software industry are faced with major challenges. While having to work efficiently using limited resources they have to perform quality assurance on their code to avoid the risk of further effort for bug fixes or compensations. Automated static analysis can reduce this risk because it promises little effort for running an analysis. We report on our experience in analysing five projects from and with SMEs by three different static analysis techniques: code clone detection, bug pattern detection and architecture conformance analysis. We found that the effort that was needed to introduce those techniques was small (mostly below one person-hour), that we can detect diverse defects in production code and that the participating companies perceived the usefulness of the presented techniques as well as our analysis results high enough to include the techniques in their quality assurance.

**Keywords:** software quality, small and medium-sized software enterprises, static analysis, code clone detection, bug pattern detection, architecture conformance analysis.

## 1 Introduction

Small and medium-sized enterprises (SMEs) play a decisive role in global software industry. In many countries, like the US, Brazil or China, these companies represent up to 85% of all software organisations [24] and carry out the majority of software development [22]. Nevertheless, SMEs are confronted with special circumstances like limited resources, lack of expertise or financial insecurity.

*Problem.* While there are many articles focusing on process improvement in SMEs [14,22,28], we found no study that looks at specific quality assurance (QA) techniques and their application in this context. Contrary to this observation, the properties of automated static analysis techniques seem to be suitable for SMEs. The benefits of such techniques lie in their low-cost application and their potential to detect critical quality defects [33,2]. Such defects are risky for the further development and increase costs. These arguments are promising for small software enterprises and their need for efficient quality assurance.

*Research Objective.* Our goal is to answer the question whether SMEs can benefit from automated static analysis techniques. Is it possible to introduce a set of such techniques in their existing projects with low effort? What kind of defects can be found using these techniques? Finally, is the perceived usefulness for the enterprises high enough to justify the needed effort? We think that these aspects are useful for future decisions in SMEs on using static analysis techniques in their projects.

*Contribution.* In this article, we describe our experience in analysing five projects of five SMEs using three different static analysis techniques: code clone detection, bug pattern detection and architecture conformance analysis. We evaluate the effort that is needed to introduce these techniques, the pitfalls we came across and how the participating enterprises evaluated the presented techniques as well as the defects we discovered in their projects.

## 2 Approach

We describe our experiences with transferring static analysis technology to small and medium-sized enterprises. This section illustrates the research context, i.e., the participating enterprises, our guiding research questions, the regarded static analysis techniques, the procedure we used to get answers to the research questions and finally the study objects we employed to gather the experiences.

### 2.1 Research Context

Fundamental for our research was the collaboration with five SMEs, all resident in the Munich area and selected through personal contacts and a series of information events and workshops. Details regarding the selection process can be found in Sec. 2.5. Following the definition of the European Commission [5], one of the participating enterprises is micro-, two are small and two are medium-sized considering their staff head count and annual turnover. The presented research is based on the experience with these enterprises gathered in a project from March 2010 to April 2011.

### 2.2 Research Questions

Our overall research objective is to analyse the transfer of new and innovative quality assurance techniques to small enterprises. We structure this objective into two major research questions.

**RQ 1.** *What problems occur while introducing and applying static analysis techniques at SMEs?*

SMEs exhibit special characteristics, such as generalist employees instead of specialists for quality assurance. Hence, smooth introduction and application are necessary so that the enterprises can adopt and make use of static analysis. We further break this down into two sub-questions:

**RQ 1.1.** *What technical problems occur?* Static analysis is tightly coupled to tools that perform and report the analysis. Hence, the ease to introduce and apply static analysis also depends on how many and which technical problems the software engineers need to solve.
**RQ 1.2.** *How much effort is necessary?*
If the effort necessary to bring the analyses up and running is too large, it can be a killer criterion for an SME, which cannot afford to reserve extra capacities for that. Therefore, we analyse the effort spent in the introduction and application.

**RQ 2.** *How useful are static analysis techniques for SMEs?*

Beyond how easy or problematic it is to introduce and apply static analysis in SMEs, we are interested in whether we can produce useful results for them. Even a small effort should not be spent if there is no return on investment. We again break this question down into two sub-questions:

**RQ 2.1.** *Which defects can be found?*
We establish a measure of usefulness by analysing the types and numbers of defects found by using the static analysis tools at the SME. If critical defects can be found, the application of the techniques is considered useful. We neither focus on specification defects and whether they can be found at all, nor do we perform cause and effects analyses for defects except for some criticality assessments.
**RQ 2.2.** *How do the companies perceive the usefulness?*
We add the subjective perception of our project partners. How do they interpret the results of the static analysis tools? Do they believe they can work with those tools and are they going to apply them continuously in their future projects? This way, we augment the information we gained from defect analysis.

## 2.3 Static Analysis Techniques

Static analysis is known as the checking of software against certain properties without executing it. It includes manual techniques, such as reviews and inspections, as well as automated techniques. As manual analyses are time-consuming and prone to missing problems in the huge amount of code to analyse, automation has high potential. For example, to detect simple and reoccurring problems in source code, such as using "==" instead of "equals" to compare strings in Java, should not be the task of human reviewers. They should concentrate on the more subtle and domain-related problems. From the interviews with our partners and the experiences at our research groups, we chose three important techniques, which we introduce in detail in the following. Technically, we employ the open-source tool ConQAT[1] for code clone detection and architecture conformance analysis as well as for results processing of bug pattern detection.

[1] `http://www.conqat.org`

*Code Clone Detection.* Modern programming languages, particularly object-oriented ones, offer various abstraction mechanisms to facilitate reuse of code fragments, but copy-paste is still a widely employed reuse strategy. This often leads to numerous duplicated code fragments—so called clones—in software systems. As stated in the surveys of Koschke [16] and Roy and Cordy [26], cloning is problematic for software quality for several reasons:

– Cloning unnecessarily increases program size and thus efforts for size-related activities like inspections and testing.
– Changes, including bug fixes, to one clone instance often need to be made to the other instances as well, again increasing efforts.
– Inconsistently performed changes to duplicated source code fragments can introduce bugs.

Code clone detection is an automated static analysis technique that focuses on finding duplicated code fragments. One of the most important metrics offered by this technique is *unit coverage*, which is the probability that an arbitrarily chosen source statement (i.e. a unit) is part of a clone. Another metric called *blow-up* denotes the ratio of the unit count of the current software w.r.t. the unit count of a hypothetical software without clones [13]. Moreover, two terms are important for clone detection: A *clone class* defines a set of similar code fragments and a *clone instance* is a representative of a clone class [12].

We differentiate between conventional clone detection and gapped clone detection. During conventional clone detection, clones are considered to be syntactically similar copies; only variable, type, or function identifiers could be changed [16]. In contrast, gapped clone detection reveals clones with further modifications; statements could be changed, added, or removed [16]. While clones are an indicator of bad design, the difference between the two approaches is that only the results of gapped clone detection can reveal defects that lead to failures, which arise through unconscious, inconsistent changes in instances of a clone class.

Clone detection is supported by a number of free and commercial tools. The most popular of them are CCFinder[2], ConQAT, CloneDR[3], and Axivion Bauhaus Suite[4]. The former two are free, while the latter two are commercial.

*Bug Pattern Detection.* By this term we refer to a technique for automated detection of a variety of defects. Bug patterns have been thoroughly investigated, e.g. in [33], and compared with other frequently used software quality assurance techniques such as code reviews or testing [31]. Bug patterns represent a scalable approach to efficiently reveal defects or possible causes thereof. Following Wagner et al. [30] they can be cost-efficient after detecting only three field defects. Their detectors, aka *rules*, aim at structural patterns recognisable from source code, executables and meta-data such as source code comments and debug symbols to

[2] http://www.ccfinder.net
[3] http://www.semanticdesigns.com/Products/Clone
[4] http://www.axivion.com

gain as much knowledge as possible from a static perspective. This knowledge encompasses obvious bugs, rather complex heuristics for latent defects, e.g. code clones (focused in Sec. 2.3), and less critical issues of coding style.

Because of the large bandwidth of defects, bug patterns are categorised along a variety of tool-specific, non-standard criteria. A reason for that is that generally applicable defect classifications are rare, vague or difficult to use in practice [29]. The tools used for this report classify their rules according to the consequences of findings such as security vulnerability, performance loss or functional incorrectness. By the term *finding* we denote that a rule was applied at a specific location. Often, findings are themselves categorised by their severity and their confidence levels.

Many of the rules are realised by means of individual lexers and parsers, by using compiler infrastructures, or by more reusable means such as pattern or rule languages and machine-learning. Rules for latent defects and coding style often stem from abstract source code metrics as, e.g., realised in Ferzund, Ahsan, and Wotawa [7]. Among the wide variety of tools [32] available for bug pattern detection, free and more popular ones are, e.g., splint[5] for C, cppcheck[6] for C++, FindBugs[7] for Java as well as FxCop[8] for C#.

*Architecture Conformance.* The phenomenon of architectural erosion is a widely documented problem [6,8,25]. Architectural knowledge erodes or even gets lost during the lifetime of a system. Accordingly, the documented and implemented architectures are drifting apart from each other. This effect leads to a downward spiralling maintainability of the system. In some cases the effort needed to re-implement the whole system becomes lower than to maintain it. To counteract this situation different approaches are used to compare the system's implementation with its intended architecture.

Passos et al. [23] identify three static concepts existing for architecture conformance analysis: *Reflexion Models (RM)*, *Source Code Query Languages (SCQL)* and *Dependency Structure Matrices (DSM).*

*Reflexion Models* as defined by Koshke and Simon [17] compare two models of a system to each other and check their conformance. The first model usually represents the intended architecture, the second one the implementation of the system [15]. The intended architecture consists of components and allowed relationships between components, expressed as rules. Each component itself can contain sub-components. The system's code is mapped to these components and then analysed for conformance to the given rules. This technique is used by the commercial tools SonarJ[9] and Structure101[10] as well as the open-source tools ConQAT and dependometer[11].

[5] http://splint.org
[6] http://cppcheck.sourceforge.net
[7] http://findbugs.sourceforge.net
[8] http://msdn.microsoft.com/en-us/library/bb429476%28v=vs.80%29.aspx
[9] http://www.hello2morrow.com/products/sonarj
[10] http://www.headwaysoftware.com
[11] http://source.valtech.com/display/dpm/Dependometer

**Table 1.** Study objects

| SO | Platform | Sources | Size [kLoC] | Business Domain |
|---|---|---|---|---|
| 1 | C#.NET | closed, commercial | ≈ 100 | Corporate controlling |
| 2 | C#.NET | closed, commercial | ≈ 200 | Embedded device maintenance |
| 3 | Java | open, non-profit | ≈ 200 | Health information management |
| 4 | Java | closed, commercial | ≈ 100 | Communal controlling |
| 5 | Java | closed, commercial | ≈ 560 | Document processing |

There are tools using *SCQL* like Semmle.QL [3] or *DSM* like Lattix [27], for the sake of brevity not further explained here. Both of these concepts rely strongly on the realisation of the system and cannot provide an architecture specification that is independent of the system's implementation [4].

### 2.4 Study Subjects and Objects

*Study Subjects.* For our investigation, we collaborate with five SMEs. These companies cover various business and technology domains, e.g. corporate and communal controlling, form letter processing as well as diagnosis and maintenance of embedded systems. Four of them are involved in commercial software development, one in software quality assurance and consulting. The latter could not provide an own software project.

*Study Objects (SO).* Following the suggestion of the partner without a software project, we instead chose the humanitarian open-source system OpenMRS[12], a development of the equally named multi-institution, non-profit collaborative. Hence, our study objects are the five software systems briefly described in Tab. 1. These software systems encompass between 100 and 600 kLoC. The developments of SOs 1 to 4 are conducted or audited by the study subjects and started at most seven years ago. The project teams contain less than ten persons. Except for OpenMRS, they are located in the Munich area. The development of SOs 1 and 2 has already been finished before our project started.

### 2.5 Procedure

This section explains milestones of our investigation (Step 1–4). It explains the starting of our research (Step 1), addresses our research questions, i.e. which data have to be collected and how to achieve that (Step 2) as well as how and under which conditions our analyses have to be carried out (Step 3–4). Steps 2 and 3 take place in terms of a single, collaborative two-week *sprint* per participating enterprise.

**Step 1: Workshops and Interviews.** We conduct a series of workshops and interviews to first convince industrial partners to participate in our project and then to understand their context and their needs. First, in an information event,

[12] `http://www.openmrs.org`

we explain the general theme of transferring QA techniques and propose first directions. With the companies that agreed to join the project, we conduct a kick-off meeting and a workshop to create a common understanding, discuss organisational issues and plan the complete schedule. In addition, the partners present a software system that we can analyse as well as their needs concerning software quality. To intensify our knowledge of these systems and problems, for each partner we perform a semi-structured interview with two interviewers and a varying number of interviewees. Both interviewers take notes and consolidate them. We then compare all interview results to find commonalities and differences. Finally, we have one or two consolidation workshops to discuss our results and plan further steps.

**Step 2: Raw Data Collection.** The *source code* of at least three versions of the study objects, e.g., major releases chosen by the companies, is retrieved for the application of the chosen techniques for RQ 1 to analyse effects over time. For bug pattern detection and architecture conformance analyses, we retrieve or build executables packed with debug symbols for each of these configurations. For architecture conformance we also need an appropriate *architecture documentation*. To accomplish this step, all partners have to provide project data as far as available, i.e. source code, build environment and/or debug builds, as well as documentation of source code, architecture and project management activities.

**Step 3: Measurement and Analysis.** We apply each technique to the gathered raw data via corresponding tool runs and inspect the results, i.e. findings and statistics. To provide answers for RQ 1 we consider problems arising and efforts spent. The tool runs enable us to derive answers for RQ 2.1. To accomplish this step, the partners have to provide support for technical questions by a responsible contact or by personal attendance at the sprint meetings. One person per technique carries out this step for all SOs. The following explains how this is accomplished:

*Code Clone Detection.* We use the clone detection feature [11] of ConQAT 2.7 for all SOs. In case of conventional clone detection the configuration consists of two parameters: the minimal clone length and the source code path. In case of gapped clone detection such gap-specific parameters as maximal allowed number of gaps per clone and maximal relative size of a gap are additionally required. Based on the experience of our group and initial experimentation, we set the minimal clone length to 10 lines of code, the maximal allowed number of gaps per clone to 1 and the maximal relative size of a gap in our analysis to 30%. After providing the needed parameters we run the analysis.

To inspect the analysis metrics and particular clones we use ConQAT. It provides a list of clones, lists of instances of a clone, a view to compare files containing clone instances and a list of discrepancies for gapped clone analysis. This data is used to recommend corrective actions. Also in a series of runs of clone detection over different versions of respective systems we monitor how several parameters (cf. Sec. 2.3) evolve in subsequent versions.

*Bug Pattern Detection.* For Java-based systems we use FindBugs 1.3.9 and PMD[13] 4.2.5. In C#.NET contexts we use Gendarme[14] 2.6.0 and FxCop 10.0. Aside from applying all rules, we choose two additional tool settings which we consider as being relevant for the SOs to gain two focused quality perspectives:

1) *Selected categories* addressing correctness, performance, and security
2) *Selected rules* for unused or poorly partitioned code and bad referencing

The tool settings are determined during preliminary analysis test runs. Categories and rules which are considered as not important – based on discussion with the partners as well as requirements non-critical to the SOs' application domains – are ignored during rule selection.

To simplify the issue of defect classification (cf. Sec. 2.3) for our investigation we only distinguish between rules for *bugs* (obvious defects), *smells* (simple to very complex heuristics for latent defects) and *pedantry* (less critical issues with focal point on coding style).

For additional and language independent metrics (e.g., lines of code without comments; code-comment ratio; number of classes, methods and statements; depth of inheritance and nested blocks; comment quality) as well as for result preparation and visualisation we apply ConQAT.

Next, we analyse the finding reports resulting from the tool runs. This step involves the filtering of findings as well as the inspection of source code to confirm the severity and confidence of the findings and to determine corrective actions. To get feedback and to confirm our conclusions from the findings we discuss them with our partners during a workshop.

*Architecture Conformance Analysis.* We use ConQAT for this technique. The procedure for each system consists of four steps:

1) Configuration of the tool with path to source code and corresponding executables of the system
2) Creation of the architecture reflexion model (cf. Sec. 2.3) based on the architectural information given by the enterprises
3) Run of the architecture conformance analysis
4) Defect analysis: Identification, discussion and classification of architectural violations

A detailed description of this ConQAT feature can be found in [4]. In summary, we use a reflexion model where dependency and hierarchy relations between components can be expressed. As a next step, we map modelled components to code parts (e.g. packages, namespaces, classes). We exclude code parts from the analysis that do not belong to the system (e.g. external libraries). Then, ConQAT analyses the conformance of the system to the reflexion model. Every existing dependency that is not allowed by the architectural rules represents a defect. Defects are visualised by the tool on the level of components and on the

[13] `http://pmd.sourceforge.net`

[14] `http://www.mono-project.com/Gendarme`

level of classes and can therefore be analysed on both high and low level. To eliminate tolerated architecture violations and to validate the created reflexion model, we discuss every found defect with the enterprise. As a last step we classify all defects together with the responsible enterprise. This allows us to group similar defects and to provide a general understanding.

**Step 4: Questionnaire.** First, we evaluate the experience of the participating enterprises regarding software quality as well as static analysis techniques. Second, we want to understand the perceived usefulness of static analysis techniques for SMEs: Do they plan to use the presented techniques in their future projects? Thus, we perform a survey on our study subjects using a questionnaire containing nine questions (Q1-9), which can be found in Appendix A. This way we contribute to RQ 2.2. The executive managers of each enterprise in their role as a representative for their company then fill out this questionnaire and we evaluate the answers. To avoid the risk of biased or too narrowly formulated answers we use both, open and closed questions.

## 3 Results

We held the information event of Step 1 of our procedure (cf. Sec. 2.5) in July 2009 and invited more than thirty SMEs of which finally 12 participated. From these companies, five committed to take part in the project. The other companies were not able to provide the necessary commitment because of schedule or budget constraints. As the first discussions were generally about improving quality assurance, it was not caused by the choice of techniques. We conducted the kick-off meeting in March 2010, the interviews between March and July and, finally, two consolidation workshops in July 2010. We did not particularly analyse their outcome for this paper. But based on these interviews and the experience of our research group, we selected the three static analysis techniques and interpreted our further results.

In the following we portrait for each technique how we contribute to the posed research questions.

### 3.1 Code Clone Detection

**RQ 1.1 – Technical Problems.** Code clone detection turned out to be the most straightforward and least complicated of the three techniques. It has, however, some technical limitations that could hinder its application in certain software projects.

A major issue was the analysis of projects containing both, markup and procedural code like JSP or ASP.NET. Since ConQAT supports either a programming language or a markup language during a single analysis, it is required to aggregate the results for both languages. To avoid this complication and to concentrate on the code implementing the application logic we took into consideration only the code written in the programming language and ignored the markup code.

**Table 2.** Efforts spent (RQ 1.2) per study object for applying each of the techniques

| **Phase** | **Work step** | **(C)lone (D)etection** | **Bug Pattern Detection** | **Architecture Conformance** |
|---|---|---|---|---|
| Introduction (configur-ation) and calibration | Analysis tools | $\leq 0.5h$ | $\leq 1h$ | $\leq 0.5h$ |
| | Aggregation via ConQAT | n/a | $\leq 0.5d$ | $\leq 0.5h$ |
| | Recalibration, $x$-times | n/a | $\leq x * 0.5h$ | n/a |
| Application (analysis) | Run analysis | $\leq 5min$ | $1min \leq . \leq 1h$ | $\leq 10sec$ |
| | Inspection of results | $\leq 1h$, more for gapped CD | $5min \leq . \leq 0.5h$ | $5min \leq . \leq 0.5h$ |

Nevertheless, it is still possible to combine the results of clone detection of the code written in both languages to get more precise results.

Another technical obstacle was filtering out generated code from the analysed code basis. In one SO large parts of the code were generated by a parser generator, viz. ANTLR. We excluded this code from our analysis using ConQAT's feature to ignore code files specified by regular expressions.

**RQ 1.2 – Spent Effort.** The effort required to introduce clone detection is small compared to the other two techniques under study. The ease of introduction of clone detection is achieved due to the minimalist configuration of the analysis which in the simplest case includes the path to the source code and the minimal length of a clone.

For all SOs it took less than an hour to configure clone detection, to get the first results and to investigate the longest and the most frequent clones. Running the analysis process itself took less then five minutes.

In case of gapped clone detection it could take a considerable amount of time to analyse if a discrepancy is intended or if it is a defect. To speed up the process ConQAT supports that the intended discrepancies can be fingerprinted and excluded from further analysis runs. An overview of the efforts can be found in Tab. 2.

**RQ 2.1 – Found Defects.** The results of conventional clone detection can be interpreted as an indicator of bad design or of bad software maintainability, but they do not point at actual defects. Nevertheless, these results give first hints, which code parts must be improved. The following three design flaws were detected in all analysed systems to a certain extent: cloning of exception handling code, cloning of logging code and cloning of interface implementation by different classes.

Tab. 3 shows the clone detection results for three versions of each SO, sorted by time. In the analysed systems unit coverage as defined in Sec. 2.3 varied between 14 and 79%. Koschke [16] reports on several case studies with unit coverage values between 7 and 23% and one case study with a value of 59%, which he defines as extreme. Therefor, the SOs 1, 3 and 5 contain normal clone

**Table 3.** Results of code clone detection

| SO | Version | Analysed Units [kUnits] | Cloned Units [kUnits] | Blow-up [%] | Unit Coverage [%] | Longest Clone [Units] | Most Clone Instances |
|---|---|---|---|---|---|---|---|
| 1 | I | 15,9 | 3,5 | 119.5 | 22.2 | 112 | 39 |
| | II | 25,3 | 5,8 | 118.9 | 23.0 | 117 | 39 |
| | III | 32,3 | 7,8 | 119.2 | 24.0 | 117 | 39 |
| 2 | I | 35,4 | 14,3 | 143.1 | 40.5 | 63 | 64 |
| | II | 41,6 | 18,9 | 150.2 | 45.4 | 132 | 47 |
| | III | 39,9 | 14,6 | 137.4 | 36.7 | 89 | 44 |
| 3 | I | 51,7 | 9,4 | 114.5 | 18.2 | 79 | 21 |
| | II | 56,8 | 8,6 | 111.2 | 15.1 | 52 | 20 |
| | III | 61,6 | 8,4 | 110.0 | 13.7 | 52 | 19 |
| 4 | I | 8,9 | 6,0 | 238.8 | 68.0 | 217 | 22 |
| | II | 22,4 | 17,3 | 309.6 | 77.6 | 438 | 61 |
| | III | 38,3 | 30,4 | 336.0 | 79.4 | 957 | 183 |
| 5 | I | 196,3 | 48,7 | 122.3 | 24.8 | 141 | 72 |
| | II | 211,3 | 53,4 | 122.7 | 25.3 | 158 | 72 |
| | III | 208,6 | 53,2 | 122.8 | 25.5 | 156 | 72 |

**Table 4.** Results of gapped code clone detection

| SO | Version | Analysed Units [kUnits] | Cloned Units [kUnits] | Blow-up [%] | Unit Coverage [%] | Longest Clone [Units] | Most Clone Instances |
|---|---|---|---|---|---|---|---|
| 1 | I | 13,3 | 3,0 | 119.9 | 22.3 | 34 | 39 |
| | II | 21,0 | 4,5 | 117.9 | 21.5 | 37 | 52 |
| | III | 27,1 | 6,0 | 117.4 | 22.1 | 52 | 52 |
| 2 | I | 24,3 | 4,6 | 116.3 | 19.0 | 156 | 37 |
| | II | 34,7 | 8,7 | 123.2 | 25.0 | 156 | 37 |
| | III | 37,1 | 9,4 | 123.7 | 25.3 | 156 | 37 |
| 3 | I | 46,7 | 12,0 | 124.4 | 18.2 | 73 | 123 |
| | II | 46,1 | 10,0 | 120.0 | 15.1 | 55 | 67 |
| | III | 49,1 | 10,0 | 118.6 | 20.5 | 55 | 64 |
| 4 | I | 7,8 | 4,5 | 192.1 | 58.6 | 42 | 34 |
| | II | 18,8 | 11,0 | 206.2 | 59.8 | 51 | 70 |
| | III | 32,2 | 19,2 | 211.1 | 59.5 | 80 | 183 |
| 5 | I | 142,3 | 29,4 | 117.4 | 20.7 | 66 | 68 |
| | II | 154,0 | 32,8 | 118.0 | 21.3 | 85 | 78 |
| | III | 151,9 | 32,7 | 118.2 | 21.5 | 85 | 70 |

rates according to Koschke. The clone rate in SO 2 is higher than the rates reported by Koschke and for SO 4 it is extreme. Regarding maintenance the calculated blow-up for each system is an interesting value. For example version III of SO 4 is more than three times bigger as its hypothetically equivalent system containing no clones. SO 4 shows that cloning can be an increasing factor over time, while SO 3 reveals that it is possible to reduce the amount of clones existing in the system code.

Cloning is considered harmful because it increases the chance of unconscious, inconsistent changes, which can lead to faults in a system [12]. These changes can be detected when applying gapped clone detection. We found a number of such changes in the cloned code fragments, but we could not classify them as defects, because we lacked the knowledge needed about the software systems. Also the project partners could not directly classify these discrepancies as defects, which confirms that gapped clone detection is a more resource demanding type of analysis. Nevertheless, in some clone instances we identified additional instructions or deviating conditional statements compared to other instances of the same clone class. Gapped clone detection does not go beyond method boundaries, since experiments showed that inconsistent clones that cross method boundaries in many cases did not capture semantically meaningful concepts [12]. This explains why metrics such as cloned units or clone coverage may differ from values observed with conventional clone detection. Tab. 4 shows the results of gapped clone detection.

**RQ 2.2 – Perceived Usefulness.** Following the feedback obtained from the questionnaire, two enterprises had limited experience with clone detection, the others did not know about it at all (Q2). Three enterprises estimated the relevance of clone detection to their projects as very high, the others estimated it as medium relevant (Q3). Concerning Q3, one stated that "clones are necessary within short periods of development." Finally, all enterprises evaluated the importance of using clone detection in their projects as medium to high and plan to introduce this technique in the future (Q5). For details see Tab. 7 and 8 in Appendix A.

## 3.2 Bug Pattern Detection

**RQ 1.1 – Technical Problems.** Following Sec. 2.3, we confirm that bug patterns are a powerful technique to gather a vast variety of information about potentially defective code. However, most of its effectiveness and efficiency is achieved through carefully done, project-specific fine-tuning of the many setscrews available.

First, the impact of findings on quality factors of interest and their consequences for the project (e.g. corrective actions, avoidance or tolerance) were difficult to determine by the tool-provided rule categories, the severity and confidence information. Based on our experience we identified the following study object characteristics this impact depends on:

– Required usage-level qualities, e.g., security, safety, performance, usability
– Required internal qualities, e.g., code maintainability, reusability
– Technologies, i.e., language, framework, platform, architectural style
– Criticality of the context the findings belong to, e.g., platform or driver code

Second, some rules exhibited many false positives, either because their technical way of detection is fuzzy or because a definitely precise finding is considered not relevant in a project-specific context. The latter case requires an in-depth understanding of each of the rules, the impacts of findings and, subsequently, a proper redlining of rules as pedantry or, actually, irrelevant. We neither measured the rates of false positives nor investigated costs and benefits thereof as our focus lay on the identification of the most important findings only.

Third, due to restricted selection and filtering mechanisms in the tools as well as a bounded view of the SOs' life-cycles, we were hindered to apply and calibrate appropriate rule selectors and findings filters. We saw that the usefulness of results is crucially influenced by the conversion of project-specific information on rule impacts into queries for rule selection and findings filtering. The tools greatly differ in their abilities to achieve this task via their graphical or command-line interfaces.

We addressed the first two issues by group discussion also with our partners and improved rule selection and findings filtering to principally avoid the findings reports to get overloaded or prone to false positives of the second kind. Also, the third issue could only be largely compensated by manual efforts. As most finding reports were quite homogeneously encoded and technically well accessible, we utilised ConQAT to gain statistical information for higher-level quality metrics as listed in Step 3 of Sec. 2.5.

**RQ 1.2 – Spent Effort.** We achieved the initial setup of a single bug pattern tool in less than an hour. This step required knowledge about the internal structure of the SO such as, e.g., its directory structure and third party code. We used the ConQAT framework to flexibly run the tools in a specific setting (Java only) and for further processing of the finding reports. Having good knowledge of this framework, we completed the analysis setup for an SO (selection of rules, adjustment of bug pattern parameters, framework setup, etc.) in about half a day.

The runs took between a minute and an hour depending on code size, rules selection and other parameters. Hence, bug pattern detection should at least be selectively included into automated build tasks. Part of the rules are computationally complex and some tools frequently required more than a gigabyte of memory. The manual effort after the runs can be split into review and recalibration. The review of a report took us a few minutes up to half an hour. Due to the short period of the life-cycle of the SOs we had insight into, we could not estimate the recalibration effort for the rule selector and the findings filter. An overview of the efforts can be found in Tab. 2.

**RQ 2.1 – Found Defects.** We conducted bug pattern analysis in three selective tool settings according to Step 3 in Sec. 2.5, but only for one version of each SO. For all SOs the filtered finding reports confirmed the defects focused or expected by these settings. Without going into the quantities and details of single findings, we summarise language-specific results:

**Table 5.** Overview of bug pattern results. Legend: Cells contain the number of findings or a maximum value, "n" ...not applicable, "." ...not noticeable, "x" ...noticeable, but PMD did not offer an appropriate way to exactly count the many findings.

| **Tool** | **Rule** | **Study Objects** | | | | | **Most affected** |
|---|---|---|---|---|---|---|---|
| **(Lang.)** | (recommendations in parentheses) | 1 | 2 | 3 | 4 | 5 | **Qualities** |
| FxCop (C#) | Empty / general exception handlers | 47 | 106 | n | n | n | Maintainability |
| | Nested use of generic types | 44 | . | n | n | n | Maintainability |
| Gendarme (C#) | Deep namespaces | 35 | . | n | n | n | Maintainability |
| | Visible constants | 18 | 338 | n | n | n | Security |
| | Extensively large classes | . | 3 | n | n | n | Maintainability |
| | Extensively long methods | . | 17 | n | n | n | Maintainability |
| | Suspicious type conversion | . | 3 | n | n | n | Correctness |
| Gend., PMD | Constructor calls overwritable method | 8 | . | x | x | x | Security, stability |
| FindBugs (Java) | Unused local variables | n | n | 142 | . | . | Maintainability |
| | Inefficient string manipulation | n | n | 46 | . | . | Performance |
| | Corrupted serialisable | n | n | 55 | . | . | Correctness |
| | Return values not validated | n | n | 30 | . | . | Correctness, sec. |
| | Access of a null pointer | n | n | . | . | 1 | Sec., stability |
| | Integer shift beyond 32 bits | n | n | . | . | 4 | Correctness |
| PMD (Java) | Empty method in abstract class | n | n | x | . | x | Maintainability |
| | Max. cyclomatic complexity ($\leq 10$) | n | n | 78 | 156 | 216 | Maintainability |
| | Extensive length / size / parameter count, too many methods / fields | n | n | . | x | x | Maintainability |
| ConQAT | Max. nested block depth ($\leq 5$) | 13 | 11 | 19 | 17 | 14 | Maintainability |

**C#** Upon the rules with highest numbers of findings, FxCop and Gendarme reported *empty exception handlers*, *visible constants*, and *poorly structured code*. There was only one consensually critical kind of findings related to correctness in SO 2, viz. unacceptable loss of precision through *wrong cast during an integer division* used for accounting calculations.

**Java** Upon the rules with highest numbers of findings, FindBugs and PMD reported *unused local variables*, *missing validation of return values*, *wrong use of serialisable*, and extensive *cyclomatic complexity*, *class/method size*, *nested block depth*, *parameter list*. There have only been two consensually critical findings, both in SO 5, related to correctness, viz. foreseeable *access of a null pointer* and an *integer shift beyond 32 bits* in a basic date/time component.

Independent of the programming language and concerning security and stability we frequently detected the pattern *constructor calls an overwritable method* in 4 of 5 SOs and found a number of defects related to *error prone handling of pointers*. Concerning maintainability the SOs exhibited *missing or unspecific handling of exceptions*, manifold *violation of code complexity metrics* and various forms of *unused code*. Details are shown in Tab. 5.

**RQ 2.2 – Perceived Usefulness.** According to the questionnaire, all of the partners considered our bug pattern findings to be medium to highly relevant for their projects (Q3). The sample findings we presented during our final workshop were perceived as being non-critical for the success of the SOs but would have been treated if they had been found by such tools during the development of these software systems. The low number of consensually critical findings correlated well with the fact that the technique was known to all partners and that most of them have good knowledge thereof and regularly used such tools in their projects, i.e. at least monthly, at milestone or release dates (Q1-2). However, three of them could gain additional education in this technique (Q4). Nevertheless, all of the enterprises decided to use bug patterns as an important QA instrument in their future projects (Q5). For details see Tab. 7 and 8 in Appendix A.

## 3.3 Architecture Conformance Analysis

**RQ 1.1 – Technical Problems.** We observe two kinds of general problems that prevent or complicate each architectural analysis: The absence of an architecture documentation and the usage of dynamic patterns.

For two of the systems there was no documented architecture available. In one case the information was missing because the project was taken over from a different organisation that was not documenting the architecture at all. They reasoned that any later documentation of the system architecture would be too expensive for their enterprise. In another case the organisation was aware that their system was severely lacking any architectural documentation. Nevertheless they feared that the time involved and the sheer volume of code to be covered exceeds the benefits. The organisation additionally argued that they are afraid of having to update the documentation within several months as soon as the next release is coming out.

In SO 2 a dynamic architectural pattern is applied, where nearly no static dependencies could be found between defined components. All components belonging to the system are connected at run-time. Thus, our static analysis approach could not be applied.

Architecture conformance analysis needs two ingredients apart from the architecture documentation: The source code and the executables of a system. This could be a problem because the source has to be compilable to analyse it. Another technical problem occurred when using ConQAT. Dependencies to components solely existing as executables were not recognised by the tool. For that reason all rules belonging to compiled components could not be analysed.

Beside these problems we could apply our static analysis approach to two systems without any technical problems. An overview of all SOs with respect to their architectural properties can be found in Tab. 6.

**RQ 1.2 – Spent Effort.** For each system the initial configuration of ConQAT and the creation of the reflexion model in ConQAT could be done in less than one hour. Tab. 6 shows the number of modelled components and the rules that were needed to describe their allowed connections. The analysis process itself finished

**Table 6.** Architectural characteristics of the study objects

| SO | Architecture | Version | Violating Component Relationships | Violating Class Relationships |
|---|---|---|---|---|
| 1 | 12 Components 20 Rules | I | 1 | 5 |
| | | II | 3 | 9 |
| | | III | 2 | 8 |
| 2 | dynamic | n/a | n/a | n/a |
| 3 | undocumented | n/a | n/a | n/a |
| 4 | 14 Components 9 Rules | I | 0 | 0 |
| | | II | 1 | 1 |
| | | III | 2 | 4 |
| 5 | undocumented | n/a | n/a | n/a |

in less than ten seconds. The time needed for the interpretation of the analysis results is of course dependent on the amount of defects found. For each defect we were able to find the causal code parts within one minute. We expect that the effort needed for bigger systems will only increase linearly but staying small in comparison to the benefit that can be achieved using architecture conformance analysis as illustrated in Sec. 2.3. An overview of the efforts can be found in Tab. 2.

**RQ 2.1 – Found Defects.** As shown in Tab. 6 we observed several discrepancies in the analysed SOs over nearly all version. Only one version did not contain architectural violations. Overall, we found three types of defects in the analysed systems. Each defect represents a code location showing a discrepancy to the documented architecture. The two analysable SOs had architectural defects which could be avoided if this technique had been applied. In the following we explain the types of defects we classified together with the responsible enterprises. The companies rated all findings as critical.

- *Circumvention of abstraction layers:* Abstraction layers (e.g. presentation layer) provide a common way to structure a system into logical parts. The defined layers are hierarchically dependent on each other, reducing the complexity in each layer and allowing to benefit from structural properties like exchangeability or flexible deployment of each layer. These benefits vanish when the layer concept is harmed by dependencies between layers that are not connected to each other. In our case e.g. the usage of the data layer from the presentation layer was a typical defect we found in the analysed systems.
- *Circular dependencies:* We found undocumented circular dependencies between two components. We consider these dependencies – whether or not documented – as defects themselves, because they affect the general principle of component design. Two components that are dependent on each other can only be used together and can thus be considered as one component, which contradicts the goal of a well designed architecture. The reuse of these components is strongly restricted. They are harder to understand and to maintain.

– *Undocumented use of common functionality:* Every system has a set of common functionality (e.g. date manipulation) which is often grouped into components and used across the whole system. Consequently, it is important to know where this functionality is actually used inside a system. Our observation showed that there were such dependencies that were not covered by the architecture.

**RQ 2.2 – Perceived Usefulness.** Following the feedback gained from the questionnaire, we observed that 4 of the 5 participating enterprises did not know about the possibility of automated architecture conformance analysis (Q1). Only one of them already checked the architecture of their systems, however in a manual way and less frequently. Confronted with the results of the analysis all enterprises rated the relevance of the presented technique medium to highly relevant (Q3). One of them stated that as a new project member it is easier to become acquainted with a software system if its architecture conforms to its documented specification. All enterprises agreed on the usefulness of this technique and plan its future application in their projects (Q5). Details of the questionnaire can be found in Tab. 7 and 8 in Appendix A.

## 4 Discussion

*General Observations.* First, we observed that code clone detection and architecture conformance analysis have been quite new to our partners as opposed to bug pattern detection which was well known. This may result from the fact that style checking and simple bug pattern detection are standard features of modern development environments. However, we consider it as important to know that code clone detection can indicate critical and complex relationships residing in the code at minimum effort. We made our partners aware of the usefulness of architecture conformance analysis, both in the case of an available architecture specification and to reconstruct such a documentation.

Second, we conclude that all of the three techniques can be introduced and applied with resources affordable for small enterprises. We assume, that except for calibration phases at project initiation or after substantial product changes the effort of readjusting the settings for the techniques stays very low. This effort is compensated by the time earned through narrowing results to successively more relevant findings. Moreover, our partners perceived all of the discussed techniques as useful for their future projects.

Third, we perceive our analyses of the study objects as successful. We found large clone classes, a significant number of pattern-based bugs aside from smells and pedantry as well as unacceptable architecture violations.

*Usage Guidelines.* During the repetitive conduct of Steps 2 and 3 of the procedure in Sec. 2.5 we gained a lot of experience in applying the chosen techniques. For their introduction and application to a new software project we consider the following generic procedure as very helpful:

1) Establish a project-specific *configuration*. This includes the choice, particularly for bug patterns, of appropriate rules aiming on relevant quality factors or just the strengthening of design or coding guidelines.
2) Define events for *measurement, findings filtering and documentation*. Filtering requires in-depth knowledge of the system and its critical components. For bug pattern detection this influences severity and confidence levels, and for architecture conformance analysis this influences the definition of allowed, tolerated, and forbidden dependencies.
3) Decide whether to *treat or tolerate* findings. This involves (i) the inspection of results and defective code, (ii) the issue of change requests for defect removal and, (iii) to assess efficiency, the documentation of efforts spent.
4) Determine whether and how defects can be *avoided* regarding lessons learned from defect treatment.
5) Strengthen *quality gates* by improved criteria, which follow patterns such as, e.g., "Clone coverage in critical code package $A$ below $X\%$ prior to any bundled feature introduction.", "No critical security errors with confidence $> Y\%$ according to tool $Z$ for any release.", or "No architecture violations originating from change sets of new features."
6) For *project control* in the context of *continuous integration*, derive statistics and trends from findings reports by a quality control dashboard such as ConQAT.

## 5 Threats to Validity

In the following, we discuss threats to the validity of our results. We structure them in internal and external validity threats.

### 5.1 Internal Validity

First, a potential threat to the internal validity is that most of the project participants had little experience with the specific tools we were applying. This could give us additional technical problems, which would not have occurred with experts. Furthermore, the efforts are probably higher. We mitigated this risk by discussions with experts and we assume that the introduction in other companies would also not necessarily be performed by experts.

Second, we did not record exact details about the efforts we spent. We rather made order of magnitude estimations only. In our context we consider this threat as small as we do not require precise analyses of these efforts including time measurement.

Third, we did not completely check whether all defects we found have caused real problems such as, e.g., critical system failures during operation or significant budget overruns. Hence, there may be false positives. We reduced this risk by detailed inspections of the defects we listed.

Fourth, the questionnaire results could be wrong, because a participant either knowingly or unknowingly gave incorrect answers. We mitigated this threat by asking participants to be careful in filling it out and at the same time assured anonymity to them.

### 5.2 External Validity

As this is an experience report on a technology transfer project, the results are inherently difficult to generalise. We had five projects of SMEs all located in Germany. We also restricted our analysis to systems realised in Java and C# and only applied specific analysis tools for it. Hence, the problems, defects, and perceptions may be particular to this context.

Nevertheless, we think that most of our experiences are valid for other contexts as well. The companies, we have collaborated with, range in their size from only several to a hundred employees. The domains they build software for differ quite strongly. Finally, the tools are all prominent examples and had been used in industrial projects before. Only the restriction to two programming languages has a strong effect as for other languages there may exist rather different tools and defects. For instance, with bug pattern detection, Ahsan, Ferzund and Wotawa [1] report that characteristics of bug patterns may be language specific.

## 6 Related Work

In this research we concentrate on applying automated static analysis techniques to enable SMEs mitigate the risk of defect-related costs. Different from our approach, the research community devotes its attention primarily to software process improvement in SMEs. There are a number of papers covering this topic.

Kautz [14] developed and used metrics to evaluate how new practices and tools for configuration and change management were affecting the software process at three SMEs. This work considers that the key to successful software measurement is to make metrics meaningful and to tailor them to a particular organisation. We confirm that observation in the context of software measurement.

Von Wangenheim et al. [28] investigated the assessment of software processes in SMEs to improve these processes. They developed MARES, a set of guidelines for conducting an ISO/IEC 15504-conforming software process assessment, focused on small companies. We perceive the usage guidelines we reported as a potential bridge between automated static analysis and more general guidelines for software process improvement.

Hofer [10] states that only 10% of the analysed SMEs in Austrian software industry believe to suffer from a lack of methods. He concludes that appropriate tool support as well as the knowledge of methods is available. On the contrary, we argue that SMEs may not be aware of many effective methods and can therefore not estimate their lack concerning these techniques.

Returning to automated static analysis techniques, to the best of our knowledge, multiple techniques have never been applied in a study in an SME context. However, there are several publications in which such techniques were investigated separately and in other contexts:

Lague et al. [18] report on application of function clone detection to a large telecommunication software system. As opposed to that, we do not limit clone detection to the comparison of functions but compare arbitrary code fragments with each other. In this work we also did not analyse large systems. Nevertheless, we came to the similar conclusion that clone detection has potential to improve software quality.

Lanubile and Mallardo [19] performed research on finding clones in web applications developed using markup and programming languages. As mentioned earlier, our approach is technically limited in analysing such software systems. Introducing a semi-automatic approach presented by Lanubile and Mallardo could remove this limitation.

Ayewah et al. [2] evaluate the accuracy and value of FindBugs findings and discuss but not solve the problem of properly filtering false positives. They use the term *trivial bugs* for what we call smells and pedantry. We confirm their conclusions on the usefulness of findings and believe that an application of bug pattern detection has to undergo calibration guided by the staff of a software project. Moreover, by answering RQ2, we contribute to Foster's, Hicks' and Pugh's [9] question "Are the defects reported by [static analysis] tools important?".

Ferzund, Ahsan and Wotawa [7] report on the effectiveness of rules for smell detection. The rules they developed are based on machine learning and source file statistics provided by static code metrics. They used training information from two software projects including bug databases. We did not address the estimation of rule effectiveness but focused on their selection and application.

Wagner et al. [30] similarly applied FindBugs and PMD to two industrial projects. They could not find defects reported from the field that are covered by bug pattern detection. However, our results show that this technique indeed captures critical defects that may eventually occur in the field.

Rosik et al. [25] conducted an industrial case study on architecture conformance with three participating software engineers. They conclude that this technique should be integrated into the software engineering process and applied continuously. We think that the procedure we presented is able to satisfy their needs, because it explicitly focuses on continuous integration.

Mattsson et al. [21] illustrate their experience in an industrial project and the huge effort that is needed to keep the architectural model in conformance with the implementation. However, they tried to reach this goal in a manual way. Our results show that automation can dramatically reduce efforts.

Feilkas, Ratiu and Juergens [6] analysed three .NET platform projects of Munich Re very similar to our procedure, but they analysed the effects of the loss of architectural knowledge. Compared to our results they report a much higher effort of about five days to apply the technique, mainly because of time consuming discussions. We think that the lower effort we are reporting is mainly caused by the fact that we were collaborating with small enterprises and experienced a lower communication overhead.

## 7 Conclusions and Future Work

In general, it is most effective to combine different QA techniques to find most of the defects [20]. This, however, comes at the efforts and costs of performing many different techniques. Particularly, SMEs have difficulties in assigning large efforts to diverse QA provisions and to training specialists for them. Automated static analysis techniques promise to be an efficient contribution to software QA, because they only require little effort for their application.

We reported our experience in applying three static analysis techniques to small enterprises: code clone detection, bug pattern detection and architecture conformance analysis. Consequently, we assessed potential barriers for introducing these techniques as well as the observations we could make in a one-year project with five German SMEs.

We found several technical problems, such as multi-language projects with single language clone analysis or false positives, but we believe that these are no major road blocks for the adoption of static analysis. Overall, the effort for introducing the analyses was small. Most techniques were set up with an effort of less than one person-hour. We found various defects, such as high levels of cloning, null pointer access, erroneous calculations or circumvention of architecture layers. In the end, our partners found all of the presented techniques relevant for inclusion into their quality assurance processes.

In our opinion static analysis tools can efficiently improve quality assurance in SMEs, if they are continuously used throughout the development process and are technically well integrated into the tool landscape. But as our research was not focused on long term observations we can not address this issue. Consequently it is an interesting area of future work to investigate the long term effects of static analyses in SMEs' software projects and their continuous integration into their development processes. Questions arising from the application of these techniques such as their long term efficiency, their inclusion into an overall QA strategy, their acceptance by developers, their application to non-code development artefacts or their effects on the daily work could then be investigated.

We will continue to work in this area to better understand the needs of SMEs and investigate our current findings.

**Acknowledgements.** We would like to thank Christian Pfaller, Bernhard Schätz and Elmar Jürgens for their technical and organisational support throughout the project. We thank all involved companies for their reproachless collaboration and assistance.

## References

1. Ahsan, S.N., Ferzund, J., Wotawa, F.: Are there language specific bug patterns? Results obtained from a case study using Mozilla. In: Proc. Fourth International Conference on Software Engineering Advances (ICSEA 2009), pp. 210–215. IEEE Computer Society (2009)
2. Ayewah, N., Pugh, W., Morgenthaler, J.D., Penix, J., Zhou, Y.: Evaluating static analysis defect warnings on production software. In: Proc. 7th Workshop on Program Analysis for Software Tools and Engineering (PASTE 2007), pp. 1–8. ACM Press (2007)
3. de Moor, O., Verbaere, M., Hajiyev, E., Avgustinov, P., Ekman, T., Ongkingco, N., Sereni, D., Tibble, J.: QL for source code analysis. In: Proc. Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007), pp. 3–16. IEEE Computer Society (2007)

4. Deissenboeck, F., Heinemann, L., Hummel, B., Juergens, E.: Flexible architecture conformance assessment with ConQAT. In: Proc. 32nd ACM/IEEE International Conference on Software Engineering, vol. 2, pp. 247–250. ACM Press (2010)
5. European Commission. Commission recommendation of May 6 2003 concerning the definition of micro, small and medium-sized enterprises. Official Journal of the European Union L 124, 36–41 (May 2003)
6. Feilkas, M., Ratiu, D., Juergens, E.: The loss of architectural knowledge during system evolution: An industrial case study. In: Proc. IEEE 17th International Conference on Program Comprehension (ICPC 2009), pp. 188–197. IEEE Computer Society (2009)
7. Ferzund, J., Ahsan, S.N., Wotawa, F.: Analysing Bug Prediction Capabilities of Static Code Metrics in Open Source Software. In: Dumke, R.R., Braungarten, R., Büren, G., Abran, A., Cuadrado-Gallego, J.J. (eds.) IWSM 2008. LNCS, vol. 5338, pp. 331–343. Springer, Heidelberg (2008)
8. Fiutem, R., Antoniol, G.: Identifying design-code inconsistencies in object-oriented software: A case study. In: Proc. International Conference on Software Maintenance (ICSM 1998). IEEE Computer Society (1998)
9. Foster, J., Hicks, M., Pugh, W.: Improving software quality with static analysis. In: Proc. 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2007), pp. 83–84. ACM Press (2007)
10. Hofer, C.: Software development in Austria: Results of an empirical study among small and very small enterprises. In: Proc. 28th Euromicro Conference, pp. 361–366. IEEE Computer Society (2002)
11. Juergens, E., Deissenboeck, F., Hummel, B.: CloneDetective – A workbench for clone detection research. In: Proc. 31th International Conference on Software Engineering (ICSE 2009), pp. 603–606. IEEE Computer Society (2009)
12. Juergens, E., Deissenboeck, F., Hummel, B., Wagner, S.: Do code clones matter? In: Proc. 31th International Conference on Software Engineering (ICSE 2009), pp. 485–495. IEEE Computer Society (2009)
13. Juergens, E., Göde, N.: Achieving accurate clone detection results. In: Proceedings 4th International Workshop on Software Clones, pp. 1–8. ACM Press (2010)
14. Kautz, K.: Making sense of measurement for small organizations. IEEE Software 16, 14–20 (1999)
15. Knodel, J., Popescu, D.: A comparison of static architecture compliance checking approaches. In: Proc. IEEE/IFIP Working Conference on Software Architecture (WICSA 2007), p. 12. IEEE Computer Society (2007)
16. Koschke, R.: Survey of research on software clones. In: Duplication, Redundancy, and Similarity in Software, Schloss Dagstuhl, Germany (2007)
17. Koschke, R., Simon, D.: Hierarchical reflexion models. In: Proc. 10th Working Conference on Reverse Engineering (WCRE 2003), p. 368. IEEE Computer Society (2003)
18. Lague, B., Proulx, D., Mayrand, J., Merlo, E.M., Hudepohl, J.: Assessing the benefits of incorporating function clone detection in a development process. In: Proc. International Conference on Software Maintenance (ICSM 1997), pp. 314–321. IEEE Computer Society (1997)
19. Lanubile, F., Mallardo, T.: Finding function clones in web applications. In: Proc. 7th European Conference on Software Maintenance and Reengineering (CSMR 2003), pp. 379–388. IEEE Computer Society (2003)
20. Littlewood, B., Popov, P.T., Strigini, L., Shryane, N.: Modeling the effects of combining diverse software fault detection techniques. IEEE Transactions on Software Engineering 26, 1157–1167 (2000)

21. Mattsson, A., Lundell, B., Lings, B., Fitzgerald, B.: Experiences from representing software architecture in a large industrial project using model driven development. In: Proc. Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI 2007). IEEE Computer Society (2007)
22. Mishra, A., Mishra, D.: Software quality assurance models in small and medium organisations: A comparison. International Journal of Information Technology and Management 5(1), 4–20 (2006)
23. Passos, L., Terra, R., Valente, M.T., Diniz, R., das Chagas Mendonca, N.: Static architecture-conformance checking: An illustrative overview. IEEE Software 27, 82–89 (2010)
24. Richardson, I., Von Wangenheim, C.: Guest editors' introduction: Why are small software organizations different? IEEE Software 24(1), 18–22 (2007)
25. Rosik, J., Le Gear, A., Buckley, J., Babar, M.: An industrial case study of architecture conformance. In: Proc. 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2008), pp. 80–89. ACM Press (2008)
26. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Technical report, Queen's University at Kingston (2007)
27. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. In: Proc. 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005), pp. 167–176. ACM Press (2005)
28. von Wangenheim, C.G., Anacleto, A., Salviano, C.F.: Helping small companies assess software processes. IEEE Software 23, 91–98 (2006)
29. Wagner, S.: Defect classification and defect types revisited. In: Proc. 2008 Workshop on Defects in Large Software Systems (DEFECTS 2008), pp. 39–40. ACM Press (2008)
30. Wagner, S., Deissenboeck, F., Aichner, M., Wimmer, J., Schwalb, M.: An evaluation of two bug pattern tools for java. In: Proc. First International Conference on Software Testing, Verification, and Validation (ICST 2008), pp. 248–257. IEEE Computer Society (2008)
31. Wagner, S., Jürjens, J., Koller, C., Trischberger, P.: Comparing Bug Finding Tools with Reviews and Tests. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005. LNCS, vol. 3502, pp. 40–55. Springer, Heidelberg (2005)
32. Wikipedia. List of tools for static code analysis — wikipedia, the free encyclopedia (2011) (accessed May 6, 2011)
33. Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J.P., Vouk, M.A.: On the value of static analysis for fault detection in software. IEEE Transactions on Software Engineering 32, 240–253 (2006)

## A Results of the Questionnaire

**Table 7.** Summary of closed answers of the questionnaire for RQ 2.2 (five results, contents and answers have been translated from German to English). Legend: ++ .. very high, + .. high, o .. medium, − .. low, - - .. very low

| **Question** | **Closed Answers** (without comments) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Q1)** Which of these static analysis techniques have you already been using in your projects? | | *daily* | *weekly* | *monthly* | *less freq.* | *never* | | |
| | Architecture conformance | 0 | 0 | 0 | 1 | 4 | | |
| | Bug pattern detection | 2 | 2 | 1 | 0 | 0 | | |
| | Clone detection | 0 | 0 | 0 | 2 | 3 | | |
| **Q2)** What is your estimate of the experience of your company in these techniques? | | ++ | + | o | − | - - | none | |
| | Architecture conformance | 1 | 2 | 1 | 1 | 0 | 0 | |
| | Bug pattern detection | 1 | 3 | 1 | 0 | 0 | 0 | |
| | Clone detection | 0 | 0 | 1 | 0 | 1 | 3 | |
| **Q3)** How do you perceive the relevance of our analysis results for your study object? | | high | | o | | low | none | |
| | Architecture conformance | 3 | | 2 | | 0 | 0 | |
| | Bug pattern detection | 2 | | 3 | | 0 | 0 | |
| | Clone detection | 3 | | 2 | | 0 | 0 | |
| **Q4)** How much education could you gain from the topics of our research project? | | much | | o | | little | none | |
| | Architecture conformance | 2 | 2 | 1 | 0 | 0 | 0 | |
| | Bug pattern detection | 2 | 0 | 1 | 1 | 1 | 0 | |
| | Clone detection | 2 | 2 | 1 | 0 | 0 | 0 | |
| **Q5)** Which of the following analysis techniques do you plan to apply at which level of priority? | | ++ | + | o | − | - - | none | *) |
| | Architecture conformance | 1 | 3 | 0 | 1 | 0 | 0 | 5 |
| | Bug pattern detection | 4 | 1 | 0 | 0 | 0 | 0 | 5 |
| | Clone detection | 0 | 2 | 3 | 0 | 0 | 0 | 5 |
| | *) application of the technique is planned | | | | | | | |

**Table 8.** Summary of open answers and comments of the questionnaire for RQ 2.2 (five results, contents and answers have been translated from German to English)

**Open Answers and Comments**

**Q1)** Architecture conformance analysis has not been used because ...

- "projects have been developed cleanly or without [need of] architecture."
- "manual inspection was carried through."
- "the prerequisites ... would have needed to be established for our projects. Manual inspection (code reviews) already takes place irregularly."
- "it was not known to us."

Clone detection has not been used because ...

- "[clones were] not known to us as a problem."
- "we did not recognise its necessity."

**Q3)** The results have been relevant because ...

- "manual [code] analysis is significantly more cost-intensive, ... clone detection is only feasible with tool support."
- "we learned about concepts, experiences and tools ... it is easier to become acquainted with [a project if its architecture conforms to its documented specification]."
- "Clones are necessary within short periods of development."

**Q5)** "The results of this research project shall be included into our internal development process."
**Q6)** *Your estimate of the current status of your organisation w.r.t. software quality:*
*Strengths:* "Seamless process for requirements QA ... regarded design guidelines for all languages used ... flexible adaptation of guidelines to customer needs ... performed QA provisions (from unit testing to selective pair programming) seem to work ... so far we only experienced high customer satisfaction ... mature in testing techniques and management."
*Weaknesses:* "No consequent QA provisions ... no systematic QA ... automation and tool usage either project specific or even left out ... still learning to apply the tools."
**Q7)** *Where do you expect the highest potential of your organisation to improve its software quality?*

- "Consequent QA provisions,"
- "integrated tools and more automation ... QA dashboard for project managers,"
- "better knowledge transfer between teams and projects,"
- "improved quality control ... backflow of QA results into development process."

**Q8)** *Your estimate of the usefulness of static analysis for your software projects:*
*Positive:* "Important", "high", "trend analyses are important", "very important, because of early and efficient defect detection ... help identify structural deficits ... ease [code] maintenance ... quality improvement starting with first build ... for internal projects better control and indication of deficits."
*Negative:* "Often not feasible in projects externally conducted at the customers'."

# BIM: A Methodology to Transform Business Processes into Software Systems

Francisco J. Duarte[1], Ricardo J. Machado[1], and João M. Fernandes[2]

[1] Departamento de Sistemas de Informação, Universidade do Minho, Portugal
[2] Departamento de Informática, Universidade do Minho, Portugal

**Abstract.** This manuscript proposes a guiding methodology to obtain a software system that supports the execution of the business processes existing within an organization. The methodology promotes the usage of business process reference models and intends to reduce the implementation time of the software systems. The methodology assumes four distinct phases and several abstraction levels and is applicable both when developing systems from scratch or in re-engineering contexts. The methodology embodies a special phase to handle the diversity of the business processes of an organization. By tailoring process reference models and by considering the characteristics of a specific organization, a proper set of business processes is derived for that organization. Then, we can obtain a suitable information system and implement its automatable parts in a software solution that can run on top of open source software frameworks. We also present four new supporting concepts to the methodology, and a summarized execution of it.

**Keywords:** BPM, BIM, business implementation methodology, software, software development process, OSS, COTS, EPF.

## 1 Introduction

For business companies, representing the real world business processes in software systems is an important issue during their development and utilization. Currently, the major software houses provide solid solutions to support in software systems the execution of business processes, usually with Commercial Off-The-Shelf (COTS) systems, like Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM). The technologies used inside those COTS systems are hidden from the users, and, in the best cases, standard interface technologies (e.g., web services) are available to interact with them. In addition, when the set of business processes implemented inside the COTS system is not sufficient to support the operation of a company, a significant effort to extend the implemented processes is required, either in new projects or in re-engineering contexts. The customization of a given COTS system requires a deep knowledge on the specific options it offers. Additionally, proficiency in very specific programming languages (e.g., ABAP in SAP R/3) or in the COTS specific business model and interfaces is usually demanded in order to support the scripting inside the COTS system or to execute a proper external use of interfaces.

To deploy COTS systems, some methodologies are proposed by software vendors or are developed by consulting companies that implement those systems. These methodologies focus on tailoring the COTS system to a specific client, and in most cases assuming that the current business operations of the client should not be deeply challenged. Thus, during a COTS implementation project, the client organization cannot be sure that the most appropriate set of business processes is implemented in the software solution, neither that the COTS implemented business processes follow world-class reference business process models. Additionally, there is no guarantee that the effort needed to upgrade a subset of the implemented business processes is manageable for the organization.

Typically, the supporting tools for a COTS system load, unload, extend, or customize business processes, but only focusing on implementation issues, rather than having a business-oriented perspective. Some platforms, like ARIS [1], provide a basis for editing the software system that implements business processes, usually following proprietary process reference models. Generically, business process implementation methodologies present the following problems:

1. Methodologies are not holistic, because that they do not always start with a process reference model and they do not always end up in implemented software;
2. Methodologies are not directed neither to use mixed vendor software components nor to use Open-source Software (OSS);
3. The time to implement a software system based on non-executable business process descriptions is long;
4. There is a lack of automation for model transformation activities;
5. It is hard to pick a preferred process reference model and use it inside an established methodology;
6. Many methodologies are proprietary and induce activities caused by the available COTS software characteristics;
7. A clear focus on the quality of the business processes content is missing. Instead, a focus on the target COTS software is observed.

According to [2], the objectives of modeling a software process are to facilitate human understanding and communication, to support process improvement and management, to provide automated guidance in performing the development process, and to automate execution support. Proposals exist, like [3], to increase the automation degree in model transformations from UML into BPEL, to mechanize BPM models [4], or to use XML to allow automation of dynamic business processes [5], but a holistic approach to transform business requirements into running software is still missing. In [6], Enterprise Service Buses (ESB) are considered as a state of the art solution for a capable and manageable integration infrastructure for web services and Service-oriented Architectures (SOA). ESBs implement a message backbone an based on open-standards designed to enable the implementation, deployment, and management of SOA-based solutions. We propose a methodology called Business Implementation Methodology (BIM) to better handle the above enumerated problems, to handle properties for processes suggested in [2], and to use state of the art software frameworks. BIM starts with generic business process reference models and ends up with running software. BIM is not attached to any proprietary software platform neither to specific notations for business processes or other process artifacts. Nevertheless, for the sake of reducing the project time, BIM encourages the usage of executable business process languages and techniques to increase the degree of automation in model transformations (e.g. 4SRS [7]). BIM also provides activities to choose and use reference models and customize them to a specific organization. BIM incorporates the idea that the quality of a business software is better if, at the proper time, quality business processes are designed and presented to the software development team. BIM also promotes the usage of OSS and software provided by different vendors.

The following are the main objectives of BIM:

- To provide a holistic approach to guide software implementations for process oriented organizations, following state of the art process reference models;
- To cope with different and mixed software technologies to support the operation of real world business processes.

In the next section, we describe related work on how to use generic process reference models inside business process organizations and we make some considerations regarding established COTS implementation processes. In Section 3, some concepts to support BIM are introduced. In Section 4, the proposed business implementation methodology is described in detail, namely its phases and transitions, its process, and the notations it supports. In Section 5, we present a running example to show the use of BIM, as well as some potential implementations of the proposed concepts. Finally, in Section 6, conclusions are presented and the future work is proposed.

## 2 Related Work

The goal of a methodology is to encourage an approach to solve a particular problem with a set of methods and techniques previously chosen [8]. A methodology can be presented as a series of phases, with associated techniques and notations [9].

The concept of process reference models arose from the idea of best practices shared among organizations. Reference models represent the business operations and internal structure of an organization. Reference models can be thought of as templates from which process models may be developed [10]. Consequently, if standard processes are to be used, standard roles to intervene with those processes can also be depicted in advance. Currently, generic business process reference models are available for organizations, like the Supply-Chain Operations Reference-model (SCOR) [11], the Information Technology Infrastructure Library (ITIL) [12], or the Enhanced Telecommunications Operations Map (eTOM) [13]. Business process reference models are standards accepted by organizations on how business processes can be properly designed and managed. Reference models are usually accompanied with recommendations, training events, books, and certifications, all providing a set of tools for implementation teams. This approach relies heavily on the human skills of the implementing team and it is not immune to project environmental conditions, like a low budget for training. After the adoption of a new set of business processes comes the need to implement most of them, quickly and with quality, in some software system. Some methodologies exist to help on the development of software systems for supporting business activities. Generic software development processes can include disciplines, like the Business Modeling discipline in the Rational Unified Process (RUP) [14], to help software developers understand and define the business context of their client in a set of process deliverables. In some of those models, notation is defined applying extensions to the Unified Modeling Language (UML) [15] to design business processes. The Business Modeling discipline provides key inputs to requirements and software design activities, namely a basis to decide which business processes will be implemented in software, or some business properties that will influence decisions related with non-functional requirements. COTS software vendors also propose methodologies, like Oracle PeopleSoft Enterprise - Rapid Start [16], Microsoft Business Solutions - Navision Rapid Implementation [17], or SAP AcceleratedSAP [18], that focus on the proper implementation of their own COTS technology, assuming that they will satisfy the clients' needs. Just considering the names of the presented methodologies, one can realize the importance of the time needed to implement a COTS system. For that reason, automation efforts, like deriving executable business processes [19], are crucial to help achieving a reduced implementation time. A complementing approach is to use iterations to provide the most needed parts of the solution as soon as possible, complemented with later additions. Also, efforts from non-profit consortia, like OASIS Business Centric Methodology [20], are focusing on the proper implementation of business processes with web services to provide ubiquitous connections among information systems and the people who use them. Considering the above mentioned reasons, it is advisable that a methodology to develop software systems supporting some set of business processes includes:

- a way to deal with standard process reference models;
- the integration of tailored business processes, as demanded by the client organization;
- the promotion of automation for its techniques;
- the capacity to obtain a quality software solution in a fast way.

As processes are becoming more and more automated, the management of processes will become automated as well [10]. In the near future, organizations must reach an internal level of maturity to allow focusing on building and improving excellent business processes, rather than focusing on the daily running problems of their processes and respective software implementations.

## 3 Proposed New Concepts

BIM introduces four new concepts to support the use of process reference models during software development projects. The first concept, called Instantaneously Available Organization (IAvO), proposes the creation of templates for detailed business processes and roles, that afterward can be used by business entrepreneurs. This proposal extends the concept of process reference model. The second concept, designated Organizational Aspect (OA), is related with orthogonal characteristics that an organization can exhibit. For instance, an organization may want to implement some quality standard that is cross-cutting to all business processes. The third one, Process Framework (PF), is the main deliverable of BIM. During the BIM life-cycle, the PF is expected to pass through a series of states. The fourth concept, Orchestrated Business Object (OBO), is the software implementation of a business entity and its associated functionalities and data, compliant with business process reference models.

### 3.1 Instantaneously Available Organizations

Everyday, new organizations are created. Everyday, entrepreneurs think they have discovered a killer product or service that will bring a huge advantage over their competitors, either because the forthcoming product is an absolute premier, or because it incorporates new features that will overwhelm the competition. This business creation willing can be truncated with organizational problems. For the entrepreneurs, the internal configuration of their own organizations as well as the interfaces with suppliers and customers, can present difficult obstacles, either because they are not properly defined or, even worst, because entrepreneurs do not have any idea of what business content is necessary to be embodied into their own organizations. The set of common practices shared among organizations, operating inside the same business contexts, is not negligible. Furthermore, an organization from a vertical market can, and should, perform benchmarking activities to the processes of other organizations, either inside the same vertical market or not, and thus embodies processes from other organizations placed in a different vertical market. For that, it seems plausible that business practices materialized in business processes may be shared among organizations. For those reasons, we propose the concept of IAvO. The main goal behind using the IAvO concept is to have out-of-the-box organizations that entrepreneurs can pick and materialize on a concrete organization. This approach can also be used when a given organization wants to redesign itself. This concept assumes that entrepreneurs and organizations reach a high maturity level such that they are "humble" enough to use standard best practices instead of only imposing their ideas for the design of the processes. Apart from the process definitions, standard roles to run the processes can also be designed. Standard roles can help entrepreneurs seeing the amount and qualifications of the human capital to be hired. Despite the business competition environment, from an information systems perspective, business processes can be shared with competition.

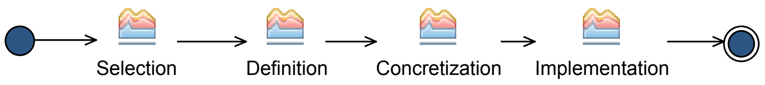

**Fig. 1.** BIM Delivery Process

This can happen because the main differentiation among competitors is based on products or core business specific activities (like a shop-floor layout), and mainly because the most decisive factor for the success of an organization is their own human capital. Inside BIM, IAvOs are considered in the first phase, where a selection of standard processes is made for later implementation. The IAvO concept differs from reference models because it is much more concrete. IAvOs share the same notations with reference models.

## 3.2 Organizational Aspects

An OA is a cross-cutting organizational concern. The concept of OA is inspired on the concept of Aspect, as introduced in Aspect-Oriented Programming (AOP) [21]. During the first BIM phase, we propose that the business analyst picks standard process reference models and, using one or more OAs, he/she creates an adequate process reference model for the organization under consideration. The process reference model incorporates a set of activities semantically compliant with some organizational standard. If some cross-cutting concern within the organization exists, then an OA can model it. With the OA concept, process reference models compliant with organizational standards, like European Foundation for Quality Management (EFQM) Excellence Model [22], can be designed or imported from the market. The OA extension mechanism, similar to that of the advices in AOP, can be a basis to use world-class added value processes without ignoring compliance with external business drivers. At the end, the surviving processes include all the desired characteristics of the chosen orthogonal quality standards. Of course, conflicts may arise, such as when creating a lean production process while maintaining traceability data, but they will be more easily tracked and explicitly solved.

Process reference models with OAs can also provide gains to organizations, when:

1. a new organization is to be created from scratch (e.g., a new subsidiary) assuring the correct transposition from mother organization of:
   (a) the local processes that may be compliant with central processes;
   (b) the standards requested locally and immediately fulfilled (e.g., social responsibility);
2. country-specific customization of processes (e.g., human resources) can be obtained from an IAvO;
3. for an existing organization, during a Business Process Re-engineering, current processes can be compared with IAvO processes that are compliant with some standard.

This last possibility provides an accurate visualization of the as-is and to-be states of the organization, and ensures that the organization can achieve the standard certification because a quantitative measure of the work is available.

### 3.3 Process Framework

A crucial concept in BIM is the PF, which is a set of process models and tools that allow the development project stakeholders to manage business processes at different states. A PF contains:

– a set of business process reference models;
– a set of allowed activities that can be performed on included process reference models;
– a set of actors that can perform activities;
– a defined state;
– a business ontology.

The PF is the main artifact being manipulated during the BIM process life cycle. To help handling the PF, visualization tools are desirable. These tools depict the PF at its different states and interpret the chosen business process description language.

### 3.4 Orchestrated Business Objects

OBOs are pieces of software that implement business entities inside some specific business ontology. They expose functionality and data to the software-implemented PF in order to allow the orchestration of business processes. Each OBO is a black-box and it should be interchangeable with other OBOs implementing the same business entity. It is highly beneficial to have a set of OBOs already available prior to the last BIM phase in order to reduce the project time and to reuse code. The concept of creating a project for developing software based on previously developed components is similar to the one proposed by the Software Factories [23] approach. The OBO concept addresses the need to have different pieces of software orchestrated to implement a specific business process, with the capacity to be interchangeable and without causing any disruption in the software system behavior.

## 4 A Business Implementation Methodology

The opportunity to generate, in an automatized way, a running software system improves the quality of the resulting product. This eliminates errors during manual model transformations and allows the developers to focus on the definitions of adequate content of business processes and software environment instead of focusing on the correctness of the models transformations. Automation also induces a reduction in the development time.

The primary goal of BIM is to guide the development of business software, ensuring the adequate support for the set of the running business processes inside an organization.

We describe BIM using to the Eclipse Process Framework (EPF) [24]. BIM is composed of four discrete time consecutive phases (Figure 1). During the first phase, a Selection of process reference models, OAs, and IAvOs is made, followed by a Definition of the chosen business processes to a particular project. Next, the Concretization of the business processes into an information system occurs. At this phase, yet there is no software implementation, and the organization may have alternative business processes defined for the same area. In the last BIM phase, one or more software Implementations are made for a subset of business processes concretized in the information system. The need to have different software implementations comes, for instance, if the organization is relying on an ERP which has annual upgrades. During those upgrade periods, the organization can benefit from having an alternative software system to overcome the inactivity of the ERP. Each BIM phase ends with a quality assessment (QA) milestone, in order to evaluate the maturity of the deliverables. The QA produces a statement expressing if the project is ready to move into the next phase (or to terminate, in the last QA). Each BIM phase follows the pattern implemented for the Selection phase activities (Figure 2). As a result, each BIM phase is composed of one or more iterations followed by a QA. While the expected quality and maturity levels of the artifacts, deliverables, and outcomes are not reached, or when some project change or constraint occur, iterations can be run incrementally inside the same BIM phase.

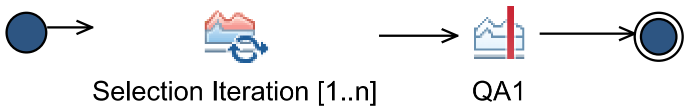

**Fig. 2.** BIM Selection Phase

During the development process, a set of state transitions must be carried on the PF, as shown in Figure 3. Starting with generic process reference models and IAvOs, an Instantiated PF is obtained. The business processes can be described, desirably in Business Process Execution Language (BPEL) [25], but also with Business Process Modeling Notation (BPMN) [26], UML, Event-driven Process Chains (EPC) [27], or Colored Petri Nets (CPN) [28]. The preference for BPEL comes from the easiness to execute it. The Instantiated PF has to consider the mission, vision, and strategic objectives, as well as other business constraints, of the organization. At this PF state, the description of the process-oriented organization is achieved, being the basis to derive the information system model. Afterward, when the PF is concretized into the information system, the PF is in the Runnable state because its processes, most likely not all, may be implemented in software. The bidirectional state transitions between the Runnable and the Software-implemented states are justified by the “loading” of the software system with a subset of the information system processes and by the "unloading" of some of those processes, becoming processes in a "vegetative" Runnable state. An ideal software implementation for the PF relies only in parametrization. There is also the possibility that some Generic PF can directly be moved into the Runnable PF state. This situation occurs when the organization does not want to make any customization of a generic process reference model or when the organization wants to fully adopt some IAvO. Desirably, the Runnable and the Software-implemented states represent the same amount of business content.

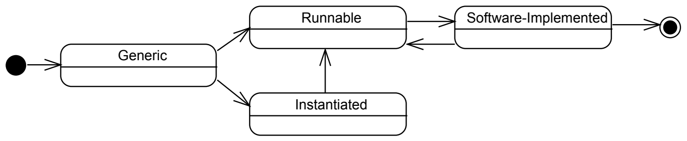

**Fig. 3.** PF States

BIM, generically summarized in Table 1, supports, during the first three phases, business process description languages. Because a software implementation is in its scope, during the last BIM phase there is a need for a computational model. In the Selection and Definition phases, activities are occurring in the domain of requirements. During the Concretization phase an information system for the organization is designed, with manual and software-implemented business processes. PF states are directly related with the BIM phases.

**Table 1.** BIM Summary

| BIM Phase | Metamodel | Domain | PF State |
|---|---|---|---|
| 1. Selection | BPEL, BPMN, EPC, CPN, ... | Business Requirements | Generic |
| 2. Definition | BPEL, BPMN, EPC, CPN, ... | Business Requirements | Instantiated |
| 3. Concretization | BPEL, BPMN, EPC, CPN, ... | Information System | Runnable |
| 4. Implementation | Computational Model | Execution | Software-implemented |

## 4.1 Activities of the BIM Phases

Inside each BIM phase, process-related functions are performed by process roles. For the Selection phase, we present the activities in Figure 4. The role Business Analyst chooses from the market an adequate process reference model (e.g., SCOR). It is advisable that he/she considers if there is available an IAvO that addresses the business processes covered by the project. The Business Analyst and the Client are responsible for defining which OAs are to be included in the PF.

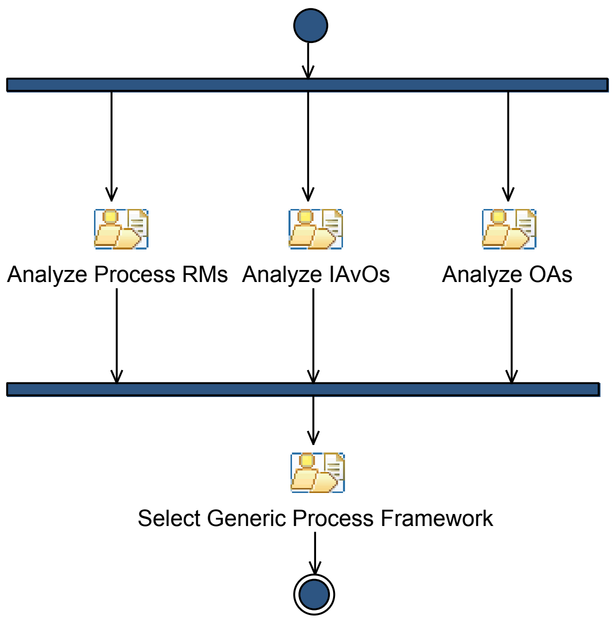

**Fig. 4.** Activities in BIM Selection Phase

The most important deliverable generated during the Selection phase is a mature and world-class generic process reference model chosen and ready to be used in the next phase. Besides the intrinsic business quality of the process reference model, its form of representation must also be chosen. Using standard process notation languages, mixed with informal graphical notation, or even using some text descriptions, can be accepted if they bring clarity to the description of the Generic process reference model that was chosen. The Generic reference model selected in the Selection phase is used to reach an Instantiated PF during the Definition phase. The Instantiated PF includes the client's view and future strategy. Business-Driven Development [29] can be used to create a proper alignment of the vision of the organization with IT tools. One key aspect of the Definition phase (see Picture 5) is the explicit utilization of an immaterial role represented by the Generic PF.

We propose that, even when the Client or the Business Process Architect do not explicitly consider world-class business processes, and for the sake of the organization, one must consider the demands of the Generic PF role in order to derive a proper Instantiated PF. The Client is not always able to decide the best solution for his/her own organization. This situation can be triggered by the ignorance of best-practices outside his/her own business environment or by the human representatives of the Client that do not always put the best interests of the organization on top of their own ones.

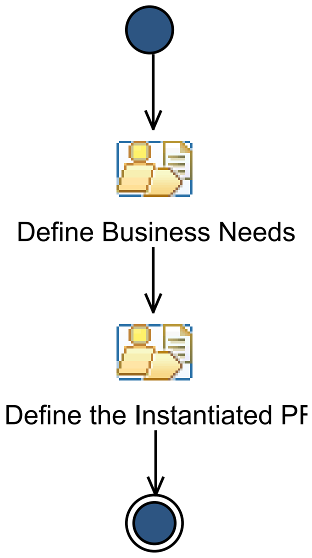

**Fig. 5.** Activities in BIM Definition Phase

After a BIM project, the complete information system, consisting both of software-implemented and manual processes, needs a regular audit activity. The PF in the Runnable state is vulnerable to end users that do not want to use the business processes of the organization, normally expressing it by creating personal worksheets and databases. Thus, we propose an audit to be carried on periodically (e.g. yearly), in order to check the compliance of the current Software-implemented PF with respect to the one designed during the BIM project. This check can only be applied if the business environment during BIM project time still applies, otherwise a new run of BIM is needed. This activity prevents changes in the Software-implemented PF without proper validations. During the audit, it is recommended to check if the manual processes are still executed the way they were planned.

Undesired changes in business process should not be confused with the ability to cope with continuous improvement activities. Whilst the former may turn the results of activities against their own host organization (e.g., the optimization of a production process that conducts to an increased product stock without a proper planning of the available space), the latter are mandatory in any organization that seeks business excellence. The goal of the third BIM phase, Concretization (see Figure 6), is the design of the Runnable PF, based on the Instantiated PF obtained in the Definition phase. The Business Process Architect role must include into the Runnable PF alternative designs for the same business processes, as well as the decisions about which business processes will have a software implementation.

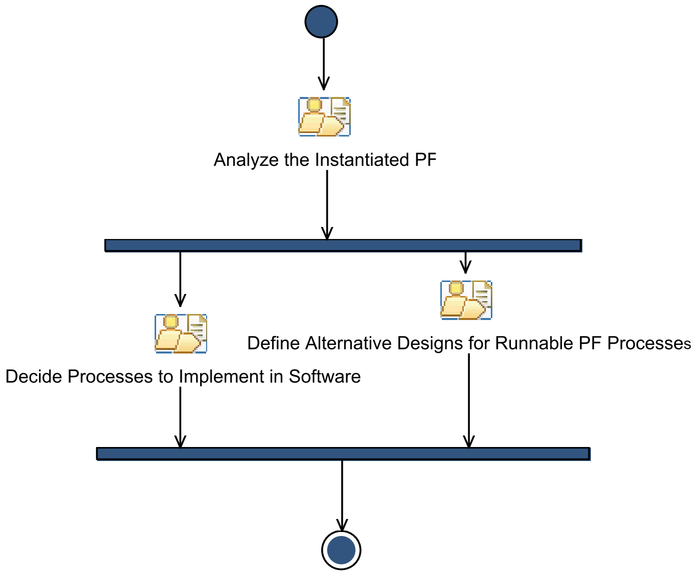
Analyze the Instantiated PF
Decide Processes to Implement in Software
Define Alternative Designs for Runnable PF Processes

**Fig. 6.** Activities in BIM Concretization Phase

Finally, during the Implementation phase (Figure 7), the Runnable PF is implemented in software. Two distinct approaches can be taken to solve the problem of generating a software solution. The first is to generate code starting with previous defined models. The second is to use already developed code, customized to cope with the requirements. With the latter approach, mixed vendor pre-developed software can be used, if compliant with the requirements of the Runnable PF. In this scenario, the client would not be tied to a particular COTS software vendor, neither he would have to rely on web services for which a proper service level agreement is unfeasible. We believe that having mixed vendor software to support business operations inside an organization brings benefits for the quality of the implemented solution, because the client can have the ability to load and unload software implementations for a part or for a complete business process. During the BIM Implementation phase, the designer of the behavior of the Software-implemented PF is the Business Process Architect and not the Software Engineer. The latter role is only concerned with providing a running software infrastructure for business processes to run. The existence of pre-developed software (OBOs) indicates that additional (outside the BIM project) software development activities may be needed and accomplished under the responsibility of the Software Engineer.

The Software-implemented PF is customizable and is the cornerstone to compare the complete OBO functionality with the Runnable PF requirements. Conflicts between the Software-implemented PF and the Runnable PF can be detected, when the latter requires more functionalities than the ones available in

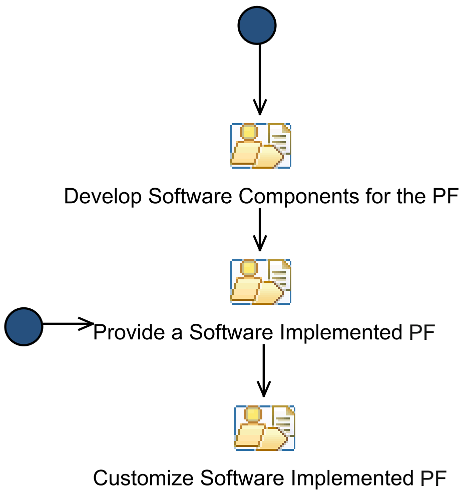
Develop Software Components for the PF
Provide a Software Implemented PF
Customize Software Implemented PF

**Fig. 7.** Activities in BIM Implementation Phase

the OBOs. This type of conflicts may demand the acquisition or the development of more OBOs, or extension of the existing ones.

### 4.2 Supporting Technology

In this section, we present a possible implementation for the Software-implemented PF, based on free software. It uses the Apache ServiceMix 4 (SMX) [30], a well-known ESB. Within an organization, a significant number of applications and platforms usually exist. This diversity requires different data formats and communication protocols to be integrated. In recent years, several technologies, such as Enterprise Application Integration (EAI), Business-to-Business (B2B), Service Oriented Architecture (SOA), and Web Services, were proposed to solve these problems. These technologies address some of the integration problems, but they are proprietary, expensive, and time-consuming to implement [31]. The ESB approach provides distributed messaging, routing, business process orchestration, transactions, reliability and security. SMX is a standard-based ESB since its design is compliant with the Java Business Integration (JBI) specification [32] and it is based on OSGi service platform [33] architecture. Since SMX is widely accepted, open source, and based on open standards, it can provide a quality, flexible, and low cost solution to implement the PF. In SMX, software components are added as OSGi bundles, like the Apache Orchestration Director Engine (ODE) [34]. ODE is a BPEL executing engine.

SMX, ODE, and BPEL, overcome the need for model transformations in the PF, since the same BPEL model can be used in all of the BIM phases. Therefore, BPEL can be used to describe the Generic, the Instantiated, the Runnable, and the Software-implemented PFs. During the BIM life-cycle desirably only the business semantics needs to be changed, maintaining the same language syntax.

Thus, one can describe, or pick from the repository of the Runnable PF, a business process in BPEL and drop it on the ODE within SMX, allowing the PF to change from the Runnable state to the Software-implemented state. In this scenario, OBOs are OSGi bundles and OBOs orchestrations are the configurations deployed into the ODE. These two implementations define a Domain Specific Language (DSL) [35] in the domain of the adopted process reference model. It means that the less abstract parts of the processes of the Generic PF must also exist in the SMX as bundles (Section 5 presents an example). During a BIM project it is assumed that all the needed OBOs (derived from the DSL of the Generic PF) are implemented as bundles in the SMX. If this is not the case, then one must acquire the needed OBOs or initiate a project to create them.

## 5 A Summarized Execution of the BIM

To demonstrate BIM, we use BPEL and SCOR. SCOR is configurable and contains descriptions and relationships among processes, as well as standard metrics to evaluate process performance. SCOR also embodies management practices and methods to reach some of its deliverables (e.g., Business Scope Diagrams) [11]. The three most abstract levels of business processes in SCOR are:

- Level 1: Top Level (Process Types). There are only five process types (Plan, Source, Make, Deliver, Return);
- Level 2: Configuration Level (Process Categories). Organizations implement their strategy by choosing different configurations to their process, like "Make-To-Order" or "Make-To-Stock";
- Level 3: Process Element Level (Decompose Processes). Organizations fine-tune their strategy by creating process element definitions, state their inputs, outputs, metrics, and pointing best practices.

In SCOR, the levels below the third one are left to organizations to implement their own specific practices so that they can cope with changing business conditions and achieve competitive advantages. In these lower levels, organizations are invited to use classic hierarchical process decomposition:

- Level 4 contains "Tasks" to implement "Process Elements";
- Level 5 contains "Activities" to implement "Tasks";
- Level 6 is where detailed actions are used to implement "Activities".

As an example, we use level 3 "D1.10 Pack Product" Process Element. It is part of the Configuration Level "Deliver Stocked Product" (level 2), and of the Top Level "Deliver" (level 1). The Tasks shown in Figure 8 represent a chosen configuration for the Process Element "D1.10 Pack Product". Each Task, like "D1.10.1 Sort Product", can be refined with another BPEL diagram.

In our example, the level 3 of SCOR will define the DSL used in all BIM phases. This means that the OBOs are all SCOR level 3 Process Elements and that they must exist inside the SMX as bundles. The internal architecture of each OBO is left open. BPEL is built with a Web Services mindset, which could prevent the mapping of manual processes during all the BIM phases. To overcome this limitation, all the process components, either with a manual or software existence, are mapped in the BPEL description. When a manual action is to be executed, its trigger comes from the software system and the result of its execution must also be informed to the software system.

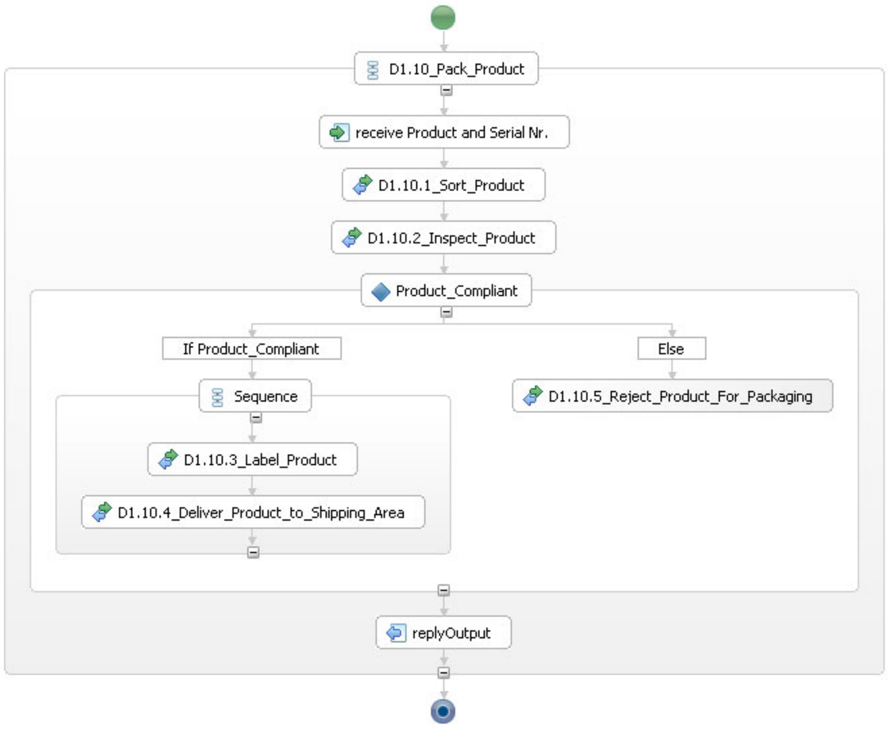

**Fig. 8.** SCOR Process Element D1.10 described in BPEL

A set of configurations of SCOR Process Elements can materialize an IAvO. After defining the Configuration Level for Top Level processes and define the appropriate Process Elements, a Business Architect can also define all Tasks, Activities, and detailed actions, thus creating an IAvO.

The degree of variability inside the IAvO is an option of its creator. It can be tracked by alternative definitions contained inside the same level of SCOR for the same process component, i.e. an IAvO can allow its users to employ “Make-To-Stock” or “Make- To-Order” strategies. If the IAvO creator wants to completely define the organization, he/she can create all the six levels of details. Depending on the business context of a process component, namely the number of goods or the time to handle them in packaging, a proper set of business roles can be drawn and incorporated into the IAvO. For instance, the IAvO’s business process described in Figure 8 implies that four business roles may exist: Product Sorter, Product Inspector, Product Labeler, and Product Deliverer. The IAvO creator can define if the business roles are to be performed by the same individual or by a set of persons. Afterward, the IAvO user can decide if the IAvO will be implemented exactly as it is defined (e.g., when a multinational organization is creating twin companies) or if some deviation is possible (e.g., when he/she is an independent entrepreneur).

To show an OA, in our SCOR-based IAvO, we consider the implementation of an Excellence Model in our organization, like the EFQM model [22]. In the EFQM model, an important part of the business activities is their reviewing. Inside an organization, one must know how well the business activities were performed, what are the causes for not achieving expected results, and how our business activities can be improved based on some measure or benchmarking. Such business concerns may be expressed at different levels of the SCOR model: if we are "lazy", we may review our processes at Process Type level; if we are "micro-managers", we may review all our level 6 detailed actions. We suggest to implement OAs at SCOR level 3, because it is the first level where concrete activities are performed. Additionally, this level is sufficient abstract and manageable to cope with changes in its Tasks, Activities, and detailed actions. For that to occur, one can decide to perform a review, every time the business process is executed. This additional Activity, the OA, can also be implemented as a bundle. The interception of the end of the Process Element D1.10 Pack Product can be done, inside the SMX, by a route in Apache Camel [36], a Spring based integration framework that implements Enterprise Integration Patterns [37].

Following, we present a summarized execution of the BIM for a goods manufacturer. In the Selection phase, the execution of the activities (see Figure 4) resulted in the the selection of the SCOR, an OA to allow reviewing all the process elements, and no IAvO considered. With these results, we selected as our Generic PF the SCOR. At this stage, a representation in BPEL of the reference model (SCOR in the case) must be available. The QA 1 is performed (see Figure 2) and the result is that the SCOR, due to its characteristics and nature, is approved as an adequate reference model.

For the Definition phase (see Figure 5), the execution of activities resulted in the identification of Quality as a major business need. The SCOR was extended by defining Tasks, Activities, and detailed actions. One extension example is showed in Figure 8 where, and resulting from the execution of D1, was included a Task "D1.10.2 Inspect Product" to guarantee high quality levels in product deliveries. The Generic PF role was instantiated with SCOR, meaning that the Instantiated PF explicitly considered it. After this second BIM phase, it is required to have the new Instantiated PF represented in BPEL. The Instantiated PF is evaluated in the QA 2.

In the Concretization phase (see Figure 6), activities execution resulted in the identification of a need for an alternative "D1.10 Pack Product" Process Element to allow the movement of products to shipping areas when no labels are present. Both alternative Process Elements "D1.10 Pack Product" will have software implementations. Due to the alternative "D1.10 Pack Product", the

Tasks “D1.10.3 Label Product” and “D1.10.4 Deliver Product to Shipping Area” (in Figure 8) will exchange their positions for the alternative design (not showed). Then QA 3 is performed.

For the BIM Implementation phase (see Figure 7), activities resulted in the customization of ODE by deploying the BPEL process definitions made for the Runnable PF. The needed “D1.10 Pack Product” OBOs are loaded into SMX. BIM ends with the the execution of QA4 to prove that there is an Software-implemented PF in the SMX running according the organization requirements.

To properly execute BIM some preconditions are needed. The most important is that the client organization is willing to improve its business processes by considering external inputs, namely in the form of process reference models. It is also critical to have human resources able to understand and use effectively process reference models, and to have skilled IT staff able to use different software frameworks. During the project phase, the project team members must have the necessary empowerment and management support to transform the business processes.

From the few executions of BIM, the main feedback from business process experts is that the usage of reference models is time consuming at the beginning due to the used notations and to the capacity needed to understand a reference model. Software experts point out that having requirements expressed by customized reference models in a clear form and early in the project increases the quality of the resulting software and shortens the time needed to implement it, mainly by avoiding cycles during the specification and validation of requirements.

## 6 Conclusions and Future Work

This manuscript presents BIM, a high-level methodology to properly implement in software a desired set of business processes. At present, BIM is designed to act as a guide using a phased development process. Also, the explicit inclusion of new concepts, like IAvO, OA, PF, and OBO, can spot the importance of the underlying activities and trigger their discussion. IAvOs allow entrepreneurs, namely the inexperienced ones, to start a business with a higher probability of success, and/or to reduce the time needed to set up the internal structure of an organization, by defining a core set of business processes and business roles. OAs can help on configuring business architectures to cope with cross-cutting business concerns, like quality standards, without significant effort. PFs can be the basis to smoothly transpose requirements and definitions from the business into the software execution domain. OBOs constitute an important concept, namely by allowing the client organization to use a mixed vendor software solutions to support its business processes. BIM can also help on considering process reference models during business process implementation projects, by embodying best-practices and thus improving business operations.

BIM is not attached to any software implementation. BIM allows the usage of different vendor software, including OSS, according to the best interest of the client and not necessarily the best interest of the software vendor. BIM promotes the usage of COTS and OSS by using already developed software frameworks, like Apache ServiceMix, namely during the Implementation phase.

Organizations must be immune to egocentric employees with “fantastic” ideas. Sometimes, proposals coming from world class processes are offered but not considered due to the pride of those employees. This behavior can be a serious obstacle to improve the internal structure and the results of the organization. BIM explicitly considers business process reference models to wide the mindset of organizations, but also considers that detailed refinements of business processes are the way for organizations to express their excellence.

BIM is a wide-scope methodology, since it deals with business definitions and also with software implementations. For that, BIM is currently proposed in a high-level of abstraction, so that it can consider several concepts of the domains it covers and it can remain independent of technologies, notations, and methods. BIM brings four main advantages over current methodologies:

– it has a holistic scope, since it starts from generic process reference models and ends in implemented software systems;
– it avoids model transformations by using the same model for business processes during all the life-cycle;
– it promotes that a software system can include components from distinct software vendors, by using the best OBOs from each vendor to compose a valuable solution for the client organization;
– it can significantly reduce the time to implement a solution, since the very same model to describe and execute business processes is used along the complete life-cycle of BIM.

BIM is adaptable to different process reference models and software solutions, it addresses both the business and the software domains with a traceable artifact, the PF, and it proposes automation.

In the future, several issues will be tackled to improve BIM: (1) develop tools to support it; (2) extend the guidance to properly tailor the methodology to a specific project taking into consideration the possible methods and techniques that can be used during a BIM project, the skills of the development team, the business context of the organization, the business requirements for the process framework, or the available implementation time; (3) to test BIM in demanding business projects; (4) to develop IAvOs, using some business process description language.

## References

1. Scheer, A.: ARIS-Business Process Modeling. Springer-Verlag New York, Inc., Secaucus (2000)
2. Curtis, B., Kellner, M., Over, J.: Process modeling. Communications of the ACM 35(9), 75–90 (1992)
3. Gardner, T.: Uml modelling of automated business processes with a mapping to bpel4ws. In: Proceedings of the First European Workshop on Object Orientation and Web Services at ECOOP 2003, Citeseer (2003)

4. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In: IEEE International Conference on e-Business Engineering, ICEBE 2005, pp. 535–540. IEEE (2005)
5. Chen, S., Chung, J., Cohen, M., Fu, S., Gottemukkala, V.: Dynamic business process automation system using xml documents, uS Patent 6,507,856, January14 (2003)
6. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer 40(11), 38–45 (2007)
7. Machado, R., Fernandes, J., Monteiro, P., Rodrigues, H.: Transformation of UML Models for Service-Oriented Software Architectures. In: The 12th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, Maryland, USA, pp. 173–182 (2005)
8. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering (1991)
9. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-oriented modeling and design. Prentice-Hall, Inc., Upper Saddle River (1991)
10. Snabe, J., Rosenberg, A., Møller, C., Scavillo, M.: Business Process Management: The SAP Roadmap. Galileo Press, SAP Press (2008)
11. Supply-Chain Council, Supply Chain Operations Reference Model (SCOR), Version 9.0 (2008)
12. OGC, Best management practice: Itil v3 and iso/iec 20000 (2008)
13. Kelly, M.: The telemanagement forum's enhanced telecom operations map (eTOM). Journal of Network and Systems Management 11(1), 109–119 (2003)
14. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Longman Publishing Co., Inc., Boston (2000)
15. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language reference manual. Addison-Wesley Longman Ltd., Essex (1998)
16. Oracle PeopleSoft, Peoplesoft enterprise - rapidstart for healthcare (2007), http://www.oracle.com/industries/healthcare/oracle-healthcare-enterprise-rapid-start-ds.pdf
17. Microsoft Navision, Automotive manufacturer deploys integrated erp solution and boosts customer service in just 15 days (May 2005)
18. Daneva, M.: Six Degrees of Success or Failure in ERP Requirements Engineering: Experiences with the ASAP Process. In: International Workshop on COTS and Product Software: Why Requirements are so Important, vol. 11 (2003)
19. Weber, I., Haller, J., Mulle, J.: Automated derivation of executable business processes from choreographies in virtual organisations. International Journal of Business Process Integration and Management 3(2), 85–95 (2008)
20. OASIS, Business-Centric Methodology (BCM), OASIS Std., Rev. 1.0 (May 2006), http://www.oasis-open.org/committees/download.php/17942/BCM.OASIS.Specification.2006-05-01.zip
21. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Longtier, J., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Auletta, V. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
22. EFQM, EFQM Excellence Model - Large Companies, Operational and Business Units version. European Foundation for Quality Management (2001), http://www.efqm.org
23. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley (2004)

24. Eclipse EPF project. Eclipse process framework (epf). Eclipse Process Framework (EPF) project (January 2011), http://www.eclipse.org/epf/general/description.php
25. OASIS, Web Services Business Process Execution Language v2.0, OASIS Std., Rev. 2.0 (April 2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf
26. White, S.: Introduction to BPMN, IBM Cooperation, pp. 2008–2029 (2004)
27. Scheer, A., Thomas, O., Adam, O.: Process Modeling Using Event-driven Process Chains. In: Process-aware Information Systems: Bridging People and Software through Process Technology, pp. 119–145. Wiley, Hoboken (2005)
28. van der Aalst, W.: Challenges in business process management: Verification of business processes using Petri nets. Bulletin of the EATCS 80, 174–199 (2003)
29. Mitra, T.: Business-driven development, IBM Developer Works (2005), http://www.ibm.com/developerworks/webservices/library/ws-bdd
30. A. S. Foundation. Apache servicemix 4.3. Apache Software Foundation (March 2011), http://servicemix.apache.org
31. A. S. Community, Apache servicemix 3.x users' guide, web page (February 2008), http://servicemix.apache.org/users-guide.html
32. Ten-Hove, R., Walker, P.: Java Business Integration (JBI) 1.0-JSR 208 Final Release, Sun Microsystems, Inc. Std. (2005), http://jcp.org/en/jsr/detail?id=208.
33. T. O. Alliance, OSGi Service Platform Core Specification 4.2, The OSGi Alliance Std. 4, Rev. 4.2 (June 2009), www.osgi.org
34. Apache Software Foundation, Apache ODE User Guide, Apache Software Foundation (2009), http://ode.apache.org/user-guide.html
35. van Deursen, A., Visser, J.: Domain-specific languages: An annotated bibliography. ACM Sigplan Notices 35(6), 26–36 (2000)
36. Apache Software Foundation, Apache Camel Manual, 2nd edn, Apache Software Foundation (2009), http://camel.apache.org/manual/camel-manual-2.0-SNAPSHOT.pdf
37. Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley Professional (2003)

# Mapping RUP Roles to Small Software Development Teams

Pedro Borges[1], Paula Monteiro[2], and Ricardo J. Machado[2]

[1] CIICESI, Escola Superior de Tecnologia e Gestão de Felgueiras do Instituto Politécnico do Porto, Portugal
`pmb@estgf.ipp.pt`
[2] Dept. Sistemas de Informação, Universidade do Minho, Guimarães, Portugal
`{pmonteiro,rmac}@dsi.uminho.pt`

**Abstract.** In the last decades the complexity of software development projects had a significant increase. This complexity emerges from the higher degree of sophistication in the contexts they aim to serve and from the evolution of the functionalities implemented by the applications However, many software corporations have a reduced dimension (micro, small or medium) which imposes a considerable constraint to the number of individuals that might be involved in each project. This limitation has obvious consequences to the individual's efficiency and effectiveness. In this paper we describe a Rational Unified Process (RUP) tailoring to simplify the number of RUP roles. With this tailoring we obtain one set of RUP roles that, without neglecting any critical role of the software development process, may easily be adopted by a small or medium software development team. In this paper, we present and justify a complete set of mapping rules between RUP roles and one possible configuration for small software development teams.

**Keywords:** RUP, small software teams, SME, RUP tailoring.

## 1 Introduction

Now more than ever, the software development industry is being put to the test, as a joint result of several stress factors. First, we have been witnessing a significant increase in the complexity inherent to software development projects, due not only to a higher degree of sophistication in the contexts they aim to serve, but also to the natural evolution of the out-of-the-box features offered by the myriad of available technologies and software systems. On the other hand, the ever growing importance of reducing time-to-market decreases the error margins, boosting the pressure applied on the teams to deliver better software in less time.

In order to react to this scenery and tip the playing field on their behalf, eastern corporations have responded by establishing partnerships with software factories based on developing countries and, in some cases, by creating their own off-shore software development centers. However, though these might be good solutions for large scale corporations and projects, they are inappropriate for some SMEs (Small and Medium Enterprises) [1], given the usually short-term nature of their projects and the considerably time-consuming specification requirements.

Since, SMEs urge for methodologies with the potential to help them cope with the challenges faced, arouse from the low level of process standardization, RUP (Rational Unified Process) [2] presents itself as a useful reference, given the wide set of roles proposed to structure software development teams. However, there's a lack of RUP configurations suited for micro (employing less than 10 people) and small companies (employing less than 50 people) [1]. This paper aims to help small scale organizations by providing them a RUP configuration that, without neglecting any critical function of the software development process, may easily be adopted during a project's execution period. In order to do so, the roles proposed by RUP will be thoroughly reviewed in order to select a much smaller subset of key participants that will inherit the duties of the suppressed roles [3].

The following sections are organized as: section 2 provides an overview of RUP tailoring approaches. Section 3 presents a RUP tailoring to SMEs, Section 4 presents a role mapping to the model presented and Section 5 presents a brief discussion about the roles accumulation. Section 6 presents the conclusions and future work.

## 2 Related Work

The Rational Unified Process (RUP) [2] is a well known software process development framework which extends the Unified Process [4] which in turn resulted from the integration and evolution of older processes such as Rational Approach [5] and Objectory Process [6].

RUP is presented as a disciplined approach for assigning tasks and responsibilities within an organization, with the aim of ensuring the production of high quality software that meets the needs of their users and in strict compliance with a predictable timetable and budget. This framework defines a set of activities, roles and artifacts, which need to be selected according to the software project. Each project is performed by a group of employees having one or more roles assigned. Each role participates in one or more activities and, as result, of their engagement in each activity one or more artifact is produced. Currently, this framework comprises more than eighty artifacts, one hundred and fifty activities and about forty roles.

Although RUP is widely used its structure lacks flexibility, and small enterprises that adopt it have to face a very long development cycle, and an "overload" of documentation when using it mechanically [7, 8]. To overcome the excess of documentation and the high cost of a long development cycle while, at same time, maintaining quality (or at least not reducing it too much), the software process must be tailored (by adding, merging and/or deleting activities, roles and artifacts).

Another set of research efforts [9-11] arose from the conclusions of a study presented in [12]. They consider that leaving the responsibility of tailoring RUP to each project context will cost too much time and resources; leading them to give to the teams, before the project start, an adapted version of RUP.

The work presented in [13] conveys a very pragmatic view about how RUP can be configured to "speed up" its adoption (of course without missing any procedural component considered essential) and thus prove the possibility of its successful adoption in SME contexts. In this way, the path followed was to perform a significant simplification of the list of artifacts to produce, followed a cost/benefit analysis of each of the artifacts provided by the methodology.

Following a completely different approach, [14] presents one RUP configuration primarily oriented to organizations that develop software in a process-oriented way, which may be appropriate for small. The authors present a set of business modeling artifacts whose production is considered essential. In [15] the authors continue this guideline, by analyzing further the business modeling artifacts, and presenting a way to set up a methodology that can incorporate procedural improvements to, thereby, enable organizations that adopt RUP to get a better ranking on the CMM scale.

According to [16], RUP is much too complex and sophisticated to be capable of being implemented as a successful practice. It is alleged that RUP does not frame in the best way the existing roles and that does not adequately involve the users during the transition phase.

The authors of [17, 18] present extensions to RUP in order to make it compliant with CMM, in particular with maturity levels 2 and 3. These works start to analyze the gaps between RUP and CMM and then they propose some activities and artifacts that will complement RUP.

Agile Methods (AM) are attempting to offer once again an answer to the impatient business community asking for lighter weight and at same time faster software development processes [19]. Some agile practices are used to change the team roles, like for instance, cross-functional and self organizing teams [19, 20].

Some studies discuss the integration of RUP and Agile Methods [21, 22]. They explain how RUP and Agile Methods can be used in conjunction. According to the author it is relatively easy to the RUP users to adopt AM practices by tailoring RUP.

RUP, CMMI and Agile Methods can also be used together [23]. The authors describe the components of requirements engineering process, and present the process compliance with CMMI. They also give some orientations to the usage of agile practices in the requirements engineering process.

There are several other research efforts that propose the simplification of RUP, by adopting tailoring techniques. However, none of them consider the organizational context existent in SMEs, namely from the roles point of view. This paper addresses this perspective.

## 3 Tailoring RUP for Constructing the Base Model

Next, we will describe the Base Model which corresponds to a RUP role simplification tailored to medium-sized companies, which essentially seek a software process that helps to design and implement solutions with high levels of quality (perceived by the customer) and to deal with the complexity inherent in projects of medium/high scale.

To achieve the proposed goals, we conducted a detailed analysis of the RUP roles in order to identify the roles considered to be essential, by satisfying at least one of three conditions. These conditions are defined in [3].

***System Analyst (C1, C2):*** Scope management is vital to the success of any project, otherwise different opinions between customers and suppliers are almost inevitable during the project execution. Therefore, the participation of system analysts is paramount since the beginning, to identify and document with the utmost rigor the requirements (functional or nonfunctional), in order to allow the supplier to correctly estimate the effort involved in performing the project.

Is extremely important that the person who performs this role has appropriate training, focusing mainly on two aspects: first, beyond basic knowledge in management, it is desirable to understand in detail the client's business domain and to perceive the real motivations and relevance of the requirements presented by the client; second, in order to develop his activities as appropriate and in accordance with the best practices, it is desirable that the person has been trained and presents some practice of requirements engineering techniques and methods (Software Requirements in SWEBOK [24]).

***User-Interface Designer (C2):*** The scope of this role intervention in a project varies according to the nature of the artifacts to be developed. Despite not being able to consider this role as a critical one, it is a fact that its best performance largely depends on a strong background in specific areas of knowledge (such as the software ergonomics). This domain cannot be considered widespread by the software engineering professionals.

***Database Designer (C2):*** Like the previous role, database designer is also considered essential to the process, mainly due to the specificity of its knowledge. Although database design techniques are mandatory in any computer science degree, usually the depth of the acquired skills is insufficient. It is desirable that the performer of this role has (at least) the following competencies: configuration and optimization of database engines; advanced knowledge on setting up indexes, views, and constraints; advanced knowledge on the implementation of triggers and stored procedures; and advanced knowledge on standardization of data models.

***Implementer (C1):*** Regardless of how good the architecture and design of a software solution is, it will not be successful without the involvement of the implementer (usually called programmers) of the sub-systems and components that support the desired functionalities.

***Integrator (C1, C2):*** In SMEs, it is usual to find several Implementers involved in project, each one engaged with a specific set of tasks. To ensure that the work runs in a smoothly and efficiently way, it is essential to have someone responsible for maintaining the implementers aware of the project context, for identifying the tasks to be undertaken and for appointing the person responsible for each one. This person must also decide how the individual tasks will be integrated and incorporated in the final outcome of the project. An example is the definition of the interfaces between the various sub-systems. Besides these technical tasks, the integrator is also responsible for the initial definition of the critical dates of the project and for developing a plan for the integration of the sub-systems, to allow the project manager to inform the client when each feature is expected to be available.

This role requires the capability to monitor the activities of each implementer to ensure to adopt measures to minimize the impact in case of failure. Appropriate training should include knowledge of human resource management, allowing the encouragement and empowerment of their work. It is desirable that the person has been trained and presents some practice in the SWEBOK knowledge areas of software engineering management and software engineering process.

***Software Architect (C1, C2):*** The performer of this role is responsible for setting the technologic foundation on which the project implementation should be based. When this context is imposed by the client, the software architect is responsible for estimating the technical risks involved and how they might be mitigated. Other responsibilities are: the definition of the skeleton of the system to be created; the characterization of the components (defining the functions and boundaries of each one); the advising on the frameworks that should be adopted. This role requires basic training in software architecture and design (SWEBOK knowledge area of software design) and also the capability of monitoring of the market trends to be aware of the most appropriate tools and frameworks for achieving the goals of a given project.

***Process Engineer (C1, C2, C3):*** It is considered essential the existence of a person mainly concerned with the management of the development process, its adaptation to the organizational context and monitoring its implementation, in order to identify and implement possible process improvements. This role requires a detailed knowledge of the adopted development process (in this case RUP).

***Project Manager (C1, C2):*** This is an important role because it has the responsibility to assume a global overview of the project through a detailed interaction with the internal and external participants. The project manager must create the conditions for the project to achieve success, by ensuring the timeliness and the fulfillment of all the undertaken commitments. To perform this role with quality it is important to have training in several areas such as: basic knowledge in management; knowledge about the client's business domain; project management methodologies (like PMBOK); negotiation skills.

***Project Reviewer (C3):*** This role cannot be considered critical to the project success and it does not require any specific skills besides the ones required for any of the other roles. However, it is important that the responsible for this role presents a good knowledge about the business domain. Due to certain responsibilities related to the verification and approval of several artifacts produced by other participants, and concerned about conflict of interests, the person that performs this role should not accumulate with another role within the project.

***Test Manager (C1, C3):*** It is essential the existence of a role whose major responsibility is to ensure the product quality by devising a plan for internal quality audits and coordinating its implementation. This is another role that should not be cumulated with other roles in particular with those roles related to the design and construction phase.

***System Tester (C1, C3):*** The implementation of the audit plan is performed by the system testers which may be entrusted with very different tasks, like the artifacts documental review and testing behavior. This is an essential role to the process.

***Course Developer (C2):*** The main concern of this role is the preparation and coordination of training. The person with this role needs competencies in the area of didactics and pedagogy as main skills to execute it.

***System Administrator (C1, C2):*** In software development projects it is extremely important to have a person focused on ensuring the satisfaction of the infrastructure needs inherent to the process, particularly regarding: the personal computers of each element of the project team, according to the specific needs of each one; servers that support the activities of the team; servers that support the testing procedures and quality assessment; servers that support the service provision to the outside. This person must have training in the following skills: practice in the SWEBOK knowledge area of software configuration; system administration; configuration and optimization of engine databases; negotiation and contracting of IT services.

## 4 Mapping RUP Roles into Base Model Roles

Rather than the 39 original roles proposed by RUP, as we can see by the stated in the previous section, it is feasible to reduce to 13 the number of the essential roles to

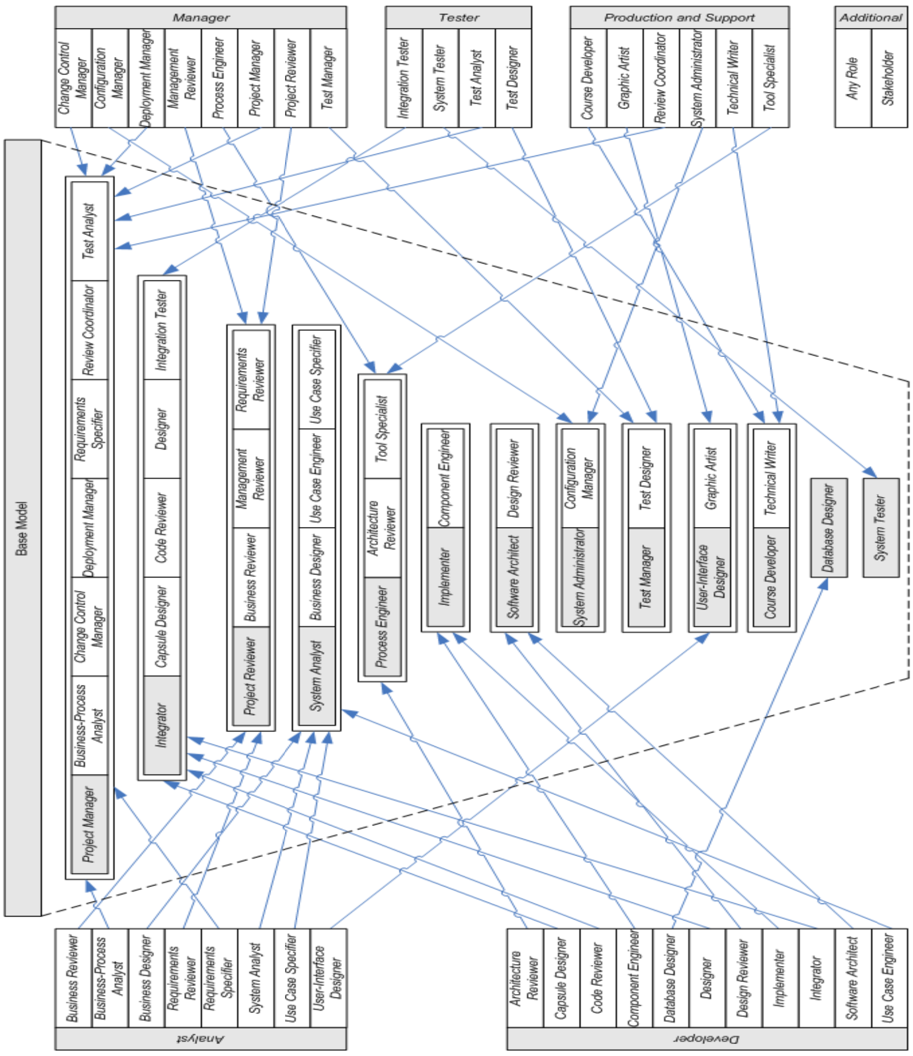

**Fig. 1.** Mappings of Base Model

implement a software process (that we call the Base Model) in the context of small development teams. However, the fact that any of the remaining 26 roles have not been regarded as essential to the process does not mean that we may discard their responsibilities or that they are not considered important for the effective and efficient implementation of the process. Instead, we propose a mapping of the remaining roles into each role previously considered essential, according to a set of guidelines defined in [3]. In figure 1, we present the Base Model roles and the mappings between RUP roles and roles considered essential in the Base Model. Next, we will present the proposed mappings and the respective justification.

***Business Reviewer, Requirements Reviewer and Management Reviewer maps into Project Reviewer:*** The *project reviewer* is characterized by RUP as someone responsible for evaluating the project artifacts on specific key moments with power and legitimacy to, if necessary, suspend its execution. Therefore, this role can only be performed by someone with a high level of responsibility and authority within the organization, possibly even at the highest level, since it is not unusual that managers of SMEs be personally involved in supervising and monitoring projects. Thus, *project reviewers* have the appropriate profile and therefore are able to evaluate the business artifacts (commercial proposals, business models, etc.) produced during the project (usually performed by the *business reviewer*).

The *requirements reviewer* has the responsibility to formally review the requirements identified and incorporate them into the use case model by the *requirements specifier*. Therefore, it is essential that this person is knowledgeable of the business domain and it should also be familiar with the modeling techniques used in the description of requirements. This role can be merged with the *project reviewer*.

The *management reviewer* and the *project reviewer* are separated by a slight difference, since both are defined as responsible for the review and evaluation of the artifacts produced at certain key moments. Thus, and assuming what was previously mentioned that the *project reviewer* must be someone with some knowledge in the business domain, the separation of these roles is not justified.

***Business-Process Analyst, Requirements Specifier, Change Control Manager, Deployment Manager, Test Analyst and Review Coordinator maps into Project Manager:*** Since the *project manager* should be the person with greater proximity to the customer, he may assume the responsibility to define the business architecture and to describe their needs (in the form of business use cases), replacing the *business-process analyst*. This would ensure that the *project manager* knows in detail the scope of the project, which allows him to deal appropriately with the requests for changes and carry out an effective monitoring of its implementation.

The *project manager* is also responsible for ensuring compliance with the scope and minimizing the contract extensions that do not provide value to the organization. However, the effectiveness of his intervention may suffer if he does not have a thorough understanding of the settled goals between the parties. For these reasons, becoming responsible for the activities of the *requirements specifier* increases his control over the project, by taking responsibility for listing and characterizing the requirements, properly managing the customer expectations and ensuring they are implemented by the project team.

In small-scale projects, it is normal that the *change control manager* responsibilities are assumed by the *project manager* or the *software architect.* However, despite being essential that the *software architect* possesses a deep and updated knowledge about the project, it is considered more important that the *project manager* ensures an effective control over it; otherwise unrealistic expectations would be created upon the client. The *project manager* can also assume such responsibilities.

The *deployment manager* plans and coordinates the transition to the user's community of the products resulting from the development efforts. The success of this type of activity largely depends on the dialogue with the client and on the planning and communication skills of the person involved. The person performing this role must work closely with the *project manager*, enough reason to merge these two roles.

The reason to map the *test analyst* into the *project manager* arises from the fact that the testing efforts must to be aligned with the project context and the needs of end users. The *project manager* is the person with more knowledge about the scope of the project and about how the artifacts will be used. In dialogue with the *integrator* and *test designer*, the *project manager* defines the scenarios that must be tested. In addition, he can also be responsible for: monitoring the progress of the testing process; analyzing the results produced by the *test designer* and ensuring that they are reported to the respective *integrators* and that they are timely corrected; evaluating the effectiveness of the testing process through the perceived quality reported by the end users; facilitating the communication between *test designer's* and *integrator's*.

Besides all the previous roles, the *project manager* is also considered to be the best role to coordinate and control the activities inherent to role *review coordinator.*

***Business Designer, Use Case Specifier, Use Case Engineer maps into System Analyst:*** The *business designer* is responsible for detailing the specification of the business solution, producing artifacts that characterize the business entities involved, their expectations and interactions. However, despite being focused on business, the activity of this role is closer to the *system analyst* responsible for the identification and documentation of the project requirements. Since, the *system analyst* must also know the business domain, we recommend merging it with the *business designer.*

The *use case specifier* role interacts closely with the end users and works together with the *system analyst* in the description of the use cases that embody the identified requirements. Since this role is not defined as having their own specific tasks but only acts as an assistant, he should be merged with the *system analyst.*

The *use case engineer* is responsible for ensuring that one or more embodiments of the use cases represent, in a coherent and comprehensive way, the project's requirements. Therefore, and given the tasks performed by the *system analyst* during requirements elicitation, we considered that the merge with the *use case engineer* activities is a natural extension of his work.

***Architecture Reviewer and Tool Specialist maps into Process:*** The *process engineer* role supports the project methodology and is responsible for monitoring its implementation and making the necessary adjustments to optimize its effectiveness.

The *architecture reviewer* is explicitly a technical role, since he formally reviews the architecture designed by the *software architect*, in order to validate the design choices. It is important that the *architecture reviewer* has the necessary legitimacy to point out mistakes or omissions. Apart from the obvious technical skills, it is important to have good communication skills, enabling to manage any conflicts with the required sensitivity and delicacy. The *process engineer* is the person in best suitable for accumulating the *architecture reviewer* responsibilities. By the nature of his function, he has the necessary legitimacy to evaluate the performance of everyone involved in the software development, on which he should have extensive experience.

In what concerns the *tool specialist* (which includes the identification of stakeholders needs regarding the tools to assist/facilitate their work and the selecting the most appropriate applications to meet their needs) we have decided to map his responsibilities into the *process engineer* role.

***Capsule Designer, Code Reviewer. Designer and Integration Tester maps into Integrator:*** The *capsule designer* has a profile similar to the *designer* but more focused on the accomplishment of the components performance requirements. Thus, given the similarity of both profiles, the *integrator* role assumes those responsibilities.

When coordinating the software development efforts of a team, the *integrator* needs a good level of technical expertise, which means that he should also be confident in assuming the responsibilities of the *code reviewer*, by reviewing and auditing the code source produced by the *implementers*. This means, the *integrator* must also verify if each component has been implemented in accordance with his instructions and detect potential problems.

The *designer* must translate the architecture conceived by the *software architect* into a coherent solution of components/modules/sub-systems to be implemented, detailing responsibilities, operations and relations between them. Taking into account that the *integrator* is responsible to ensure a successful integration of several existing components, it makes sense that both roles are performed by the same person, maximizing the efforts involved in the design stage.

Considering that the primary responsibility of the *integration tester* is to perform the integration tests, which are essential to verifying that the various components that make up the solution are working well together, this role can be considered as a natural extension of the activities conducted by the *integrator*.

***Component Engineer maps into Implementer:*** The *component engineer* is focused on the design of the internal structure of each sub-system, particularly regarding with operations, methods, attributes, relationships and requirements of each design class. This role is strictly related with the *implementer's* duties.

***Design Reviewer maps into Software Architect:*** The RUP methodology strongly suggests the existence of roles especially devoted to review artifacts written by third parties. Therefore, in order to maintain the independence, we suggest that the role of the *design reviewer* is accumulated by the *software architect*.

***Configuration Manager maps into System Administrator:*** We suggest merging the responsibilities of the *configuration manager* and the *system administrator*. If the *system administrator* is already responsible for managing and providing the infrastructure used by the project team, it makes sense to also perform the configuration management.

***Test Designer maps into Test Manager:*** From the activities carried out by the *test analyst* emanates a set of test scenarios that constitute a starting point for the work to be performed by the *test designer*, which is responsible for coordinating the planning, design and implementation of the necessary tests. However, these activities are a consequence of the responsibilities that the literature attributes to the *test manager*, described as the main role responsible for the success of the testing effort. Therefore, the *test manager* must assume the test designer role.

***Graphic Artist maps into User-Interface Designer:*** The performance of the *graphic artist* benefits from a refined aesthetic sensibility and some experience in the use of image manipulation applications. However, since it is common that these characteristics are also presented in the *user-interface designer*, this role can also be the responsible for meeting the project needs of image and graphic communications.

***Technical Writer maps into Course Developer:*** Given the restrictions usually found in smaller organizations, the *implementer* role could assume the responsibilities of the *technical writer*. However, the RUP methodology suggests that the production of content support (user manuals, etc.) could be performed by different persons from those involved in the technical execution of the project. Since the activities of the *course developer* can be regarded as complementary to those of the *technical writer* (the training material he produces is also directed to end users) we recommended the merging of these two roles.

**Table 1.** Roles Accumulation Restrictions (Inside Project)

| Should not accumulate with... / If it is... | Project Manager | Integrator | Project Reviewer | System Analyst | Process Engineer | Implementer | Software Architect | System Administrator | Test Manager | User-Interface Designer | Course Developer | Database Designer | System Tester |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Project Manager | ▨ | x | x | | | | | | | | | | |
| Integrator | x | ▨ | | | | | | | x | | | | ? |
| Project Reviewer | x | | ▨ | | | | | | | | | | |
| System Analyst | | | | ▨ | | | | | | | | | |
| Process Engineer | | | | | ▨ | | | | x | | | | |
| Implementer | | | | | | ▨ | | | x | | | | ? |
| Software Architect | | | | | | | ▨ | | x | | | | ? |
| System Administrator | | | | | | | | ▨ | | | | | |
| Test Manager | | x | | | x | x | x | | ▨ | x | | x | |
| User-Interface Designer | | | | | | | | | x | ▨ | | | ? |
| Course Developer | | | | | | | | | | | ▨ | | |
| Database Designer | | | | | | | | | x | | | ▨ | ? |
| System Tester | | ? | | | | ? | ? | | | ? | | ? | ▨ |

## 5 Accumulation of Roles

After we have justified, in general, the Base Model role set it is relevant to analyze the eventual restrictions to the accumulation of several roles by the same team member. This recommendation may result in the mobilization of a higher number of resources to perform a project. However, it is also a fact that it could contribute to a better performance of each role, while avoiding ethical incompatibilities. In table 1, we present the identified restrictions of roles accumulation inside the same project.

The symbol “x” indicates an absolute restriction, which means that this accumulation must be avoided. For instance, the integrator should not accumulate with the test manager role because there are some ethical issues involved, since the test manager evaluates the artifacts produced under the integrator’s supervision. The symbol “?” indicates a conditional restriction; i.e., an accumulation of roles that might be possible if some regards are considered. As an example, the integrator presents one conditional restriction to accumulate the system tester role, because this accumulation will only be possible in the cases where the system tester tasks are not performed within the same software development line under coordination as integrator. The detailed explanation of these restrictions will be presented in a future work.

## 6 Conclusions

The Rational Unified Process is a comprehensive software development process, which aims to help organizations to efficiently use resources at their disposal to ensure the effective implementation of the goals they want to achieve. However, the lack of an appropriate RUP configuration for SMEs (small and medium sized companies) that develop software has justified our effort to propose a reduced set of roles involved in the implementation of the RUP methodology. As a result, we have suggested the Base Model, which is a tailoring approach of RUP composed by 13 roles. The other 26 RUP roles not considered in the Base Model have been mapped into the Base Model roles according a set of presented guidelines. In this study we do not discuss how one person can handle all the activities of each performed role, in a reasonable time. However this is a relevant issue and it will be analyzed in a future work.

As future work, we will develop a Reduced Model that will simplify further the set of RUP roles to be adopted in small settings. This simplification will be performed by considering part of the Base Model roles as non essential and the other part as essential. The roles considered as non essential are mapped into the essential roles. We will assess the Reduced Model with a case study of a CMMI level 3 certified team [25, 26].

## References

1. European Commission. Small and Medium-sized Enterprises Definition, vol. 2011 (2005)
2. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley (2003)
3. Borges, P., Monteiro, P., Machado, R.J.: Tailoring RUP to Small Software Development Teams. In: 37th Euromicro Conference, SEAA 2011 (2011)
4. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley (1999)
5. Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., Houston, K.: Object-oriented analysis and design with applications, 3rd edn. Addison-Wesley (2007)
6. Jacobsen, I.: Object Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)

7. de Barros Paes, C.E., Hirata, C.M.: RUP Extension for the Development of Secure Systems. In: ITNG 2007, pp. 643–652 (2007)
8. Jieshan, L., Mingzhi, M.: A Case Study on Tailoring Software Process for Characteristics Based on RUP. In: CiSE, pp. 1–5 (2009)
9. Hanssen, G.K., Westerheim, H., Bjørnson, F.O.: Tailoring RUP to a Defined Project Type: A Case Study. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 314–327. Springer, Heidelberg (2005)
10. Westerheim, H., Hanssen, G.K.: The introduction and use of a tailored unified process - a case study. In: 31st EUROMICRO Conference, SEAA 2005, pp. 196–203 (2005)
11. Hanssen, G.K., Bjørnson, F.O., Westerheim, H.: Tailoring and Introduction of the Rational Unified Process. In: Abrahamsson, P., Baddoo, N., Margaria, T., Messnarz, R. (eds.) EuroSPI 2007. LNCS, vol. 4764, pp. 7–18. Springer, Heidelberg (2007)
12. Hanssen, G.K., Westerheim, H., Bjørnson, F.O.: Using Rational Unified Process in an SME – A Case Study. In: Richardson, I., Abrahamsson, P., Messnarz, R. (eds.) EuroSPI 2005. LNCS, vol. 3792, pp. 142–150. Springer, Heidelberg (2005)
13. Hirsch, M.: Making RUP agile. OOPSLA 2002 Practitioners Reports (2002)
14. Fernandes, J.M., Duarte, F.J.: A reference framework for process-oriented software development organizations. Software and Systems Modeling 4, 94–105 (2005)
15. Duarte, F.J., Fernandes, J.M., Machado, R.J.: Business Modeling in Process-Oriented Organizations for RUP-based Software Development. In: Reference Modeling for Business Systems Analysis, pp. 98–117. Idea Group Publishing (2006)
16. Hesse, W.: Dinosaur meets Archaeopteryx? or: Is there an alternative for Rational's Unified Process? Software and Systems Modeling 2, 240–247 (2003)
17. Manzoni, L.V., Price, R.T.: Identifying extensions required by RUP to comply with CMM levels 2 and 3. IEEE Transactions on Software Engineering 29, 181–192 (2003)
18. Chang, G.: Modifying RUP to comply with CMM levels 2 and 3. In: ICISE 2010, pp. 1–5 (2010)
19. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile Software Development Methods: Review and Analysis. Technical Research Centre of Finland, Espoo, Finland (2002)
20. Ana Sofia, C.M., Felipe, S.F.S., Arnaldo, D.B.: Mapping CMMI Project Management Process Areas to SCRUM Practices. In: Proceedings of the 31st IEEE Software Engineering Workshop, pp. 13–22. IEEE Computer Society (2007)
21. Ambler, S.W.: Agile Modeling and the Rational Unified Process (RUP), vol. 2011 (2001)
22. Ambler, S.: Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. John Wiley & Sons, Inc., New York (2002)
23. Cintra, C.C., Price, R.T.: Experimenting a Requirements Engineering Process Based on Rational Unified Process (RUP) Reaching Capability Maturity Model Integration (CMMI) Maturity Level 3 and Considering the Use of Agile Methods Practices. In: Alencar, F.M.R., Sanchez, J., Werneck, V. (eds.) Workshop em Engenharia de Requisitos, Rio de Janeiro, pp. 153–159 (2006)
24. Abran, A., Bourque, P., Dupuis, R., Moore, J., Tripp, L.: Guide to the Software Engineering Body of Knowledge - SWEBOK. IEEE Press (2004)
25. Monteiro, P., Machado, R.J., Kazman, R.: Inception of Software Validation and Verification Practices within CMMI Level 2. In: Fourth International Conference on Software Engineering Advances, ICSEA 2009, pp. 536–541 (2009)
26. Monteiro, P., Machado, R.J., Kazman, R., Henriques, C.: Dependency analysis between CMMI process areas. In: Ali Babar, M., Vierimaa, M., Oivo, M. (eds.) PROFES 2010. LNCS, vol. 6156, pp. 263–275. Springer, Heidelberg (2010)

# Scaling Software Development Methods from Co-located to Distributed

Harald Klein[1], Eric Knauss[2], and Andreas Rausch[3]

[1] Siemens AG, Bunsenstrasse 43, 91058 Erlangen, Germany
h.klein@siemens.com
[2] Software Engineering Group, Leibniz Universität Hannover, Welfengarten 1, Hannover, Germany
eric.knauss@inf.uni-hannover.de
[3] Institute of Software Systems Engineering, University of Technology Clausthal, Julius-Albert-Str. 4, Clausthal-Zellerfeld, Germany
andreas.rausch@tu-clausthal.de

**Abstract.** Software projects nowadays are typically sourced from more than one location. This dispersed situation requires a higher degree of regulation mechanisms than provided in agile development methods. Workarounds for scaling agile practices to the distributed development scenario exist, which are mostly not of any value for decision makers, since they still merely provide an ad-hoc way of setting up distributed software development projects. Especially smaller distributed software projects are in demand for methodical support for this task. We propose a systematic approach – called "Collaborative Pattern Approach" – for deriving a distributed development process from an existing co-located process. Our approach focuses on a) defining cross-location collaboration and b) assessing the quality of the derived distributed development process. We demonstrate our approach in an example case.

**Keywords:** Collaboration, Software Development Methods, Distributed Software Development, Distributed eXtreme Programming, Collaboration Patterns.

## 1 Introduction

Today, many engineering products include some software that significantly contributes to the products' functionality [15]. In order to stay globally competitive and to meet requirements from customers, software has to be highly sophisticated, which demands high competencies and special resources. Many organizations gather these competencies and resources from all over the world by linking specialized departments or employing subcontractors. Thus, we observe that organizations create more and more value in a dispersed development setting [19], [20].

In dispersed teams it is even more important to have supportive processes defined than in organizations that work in one location. It is crucial for project managers to clearly define artifacts to be developed and to assign unique responsibilities.

Furthermore, the work order of artifacts is also a major issue since personal ad hoc team meetings for coordination are not possible due to the intermediate distance. But it is very challenging to define a collaborative process from scratch that incorporates all desires from every organization that takes part in a collaborative and distributed project. There are approaches that take existing software development methods and apply them to distributed software development. For example, the successful application of eXtreme Programming (XP) in distributed software development has been reported in [10], [18]. However a systematic approach how one can derive a distributed development method from existing co-located development methods is still missing.

For this reason this paper shows a structured pattern-based approach that allows for defining a collaborative process in distributed development environments. Such collaborative processes are depending on the distributed development scenario. In previous work, it was shown how this approach supports process definition, when entire process activities are delegated to peers. In this work we investigate the impact of the approach on the most difficult scenario: Collaborative execution of a single process activity (i.e. implementation) by dispersed peers [2], [6]. This approach is applied in an agile development environment. Furthermore we illustrate the benefits or drawbacks of our approach using a realistic example with a focus on the following research questions:

- How is a collaborative process in an agile project setup defined?
- What are assets and drawbacks in terms of performance of such a collaborative process?

A major agile method we used is XP that demands for a high level of communication and interaction [6], [14]. We used the collaboration pattern approach [12] to design synchronization points between dispersed developers, to decide which communication infrastructure should be used and to control the performance of the developers. The goal is to incorporate some of the advantages of XP in our distributed development process:

- High developer commitment because of shared responsibility.
- Low Risk of Failure and low amount of rework because of good customer interaction.

Our findings suggest that this could be achieved. However, a balanced costs-benefit-ratio of a development method can make the difference between success and failure. Therefore, we discuss how to assess the value of a derived distributed development process.

## 2 Related Work

As our example case addresses a distributed XP project, related work from an agile context is focused in the same way like from distributed development to illustrate the difficulties. Generally, Bird et al. show in [4] that it is possible to achieve comparable software quality in distributed development as in co-located software development.

As basis for this work we used Cockburn [6] describing that typical reason for distributed software development are based on either economical, organizational, or strategically considerations.

Intentionally, agile distributed development comes up with discrepancies, which has been investigated by Ramesh et al. [18]. They state that development processes have to be continuously improved; additionally, project managers have to improve communication, facilitate knowledge sharing, and build trust in the distributed team, as a fundamental principle for collaboration in general and valuable for especially setting up this work. The agile development principles which our example case is based upon are described by Beck in [3].

The applicability of four XP practices (Planning Game, Pair Programming, Continuous Integration, and Onsite-Customer) in distributed environments is illustrated by Kircher et al. [10]. They describe consequences for these practices when used in a distributed environment and give anecdotal evidence of the approach. However, no author focuses on processes that support collaboration by keeping each organization's development approach.

The collaborative approach in this work is based on process patterns, which are used to connect different organization's processes. This complements with the work of Braithwaite and Joyce [6] who also identified basically patterns of behavior for collaboration in distributed XP. Those are Balanced Sites, Distributed Standup, Multiple Communication Modes, and One Team, One Codebase. However, they give no information about benefit and costs of these patterns.

Our approach takes also business process model characteristics into account inspired by R. Prikladnicki et al. who address business strategies and models regarding distributed software development on (project) management level [17]. By means of a case study they focus on the difference and challenges of shared services, onshore/offshore outsourcing, and offshore insourcing. Furthermore, Meyer B. [15] defines a development model which divides the entire software system in sub-projects representing components of the system. This generates another process with little overlaps, which requires an intensive amount of coordination and communication.

Communication is also one of the essentials in distributed development. Wolf et al. were able to give objective evidence of the importance of good communication [23]. Based on analysis of social networks they are able to predict failures in the integration builds of a large highly distributed software project. Additionally, Herbsleb and Mockus have investigated speed and communication in globally distributed software development [6] using data from a change management system and a survey performed in two organizations. A key finding is that distributed work items take up to 2.5 times longer than similar co-located work items. However, they do neither describe the development method defined nor how the distributed development method has been applied.

We considered also practical experience and illustration of how to connect different development processes in a distributed environment from Avritzer [1] and Wichmann [22]. However, Wichmann does not consider and describe any team structure or role model in his approach.

## 3 Context: Co-located Agile Development

Our example case investigates distributed agile software development [3] in a students' project. The project was held as a joint course by the Technische Universität Clausthal (TUC) and the Leibniz Universität Hannover (LUH). A similar, non-distributed course was held at LUH many times before. One of the main goals in the non-distributed agile course was to teach XP in a realistic environment. Our students should learn how the agile practices affect software development. Therefore, we tried hard to implement all twelve practices proposed by Beck [3]. An essential part of the course is a one week block encompassing most of the software development activities. Thus, we can implement practices like Onsite-Customer, 40h-week, and Pair-Programming. For teaching purposes, we have very short iterations. In our experience, two day iterations work best [21].

Fig. 1 describes the development process in our co-located XP classes. A major concept is story cards. These handwritten slips of paper contain a short description of how a future user will use the system that is still under development. Often, narratives (i.e. usage stories) are used. Onsite Customers write these stories together with the agile team. New story cards can always be created and added to the Product Plan. The Product Plan contains all story cards the team and customer are aware of. In our experience, most stories are created at the end or beginning of an iteration.

Each iteration starts with a Planning Game. If the customer has new stories, these are written first. After that, each story card is estimated by the developers. Then, the customer prioritizes the story cards by sorting them. The most important story card lies on top of the stack. Based on experience from past iterations customer and team select as many story cards from this stack as one iteration allows to implement. This is the iteration plan.

After the Planning Game, which is conducted in conjunction with the customer the story cards from the iteration plan are implemented via Pair Programming. Any pair of two developers takes the topmost story card ("Select Story Card") and implements it by applying the test-first practice. The developers start with writing an automatic unit test ("write unit test"), then add just enough code to make the test run ("write code"), before writing the next test. This is depicted in the activity "Pair Programming: Implementation and Integration". As illustrated with turning arrows improvement loops are included.

When the developer pair thinks that a story card is implemented, they go to the onsite customer and present their results. If these results are acceptable for the customer ("acceptance test"), the developer pair integrates them into the system and the story card is finished ("integrate story"). The pair takes the next story from the iteration plan and starts again. This iterative process loop ends if all story cards of iteration plan are processed and/or acceptance from the customer is gained.

Prior to participation of the joint course TUC followed a traditional waterfall development process which is depicted in Fig. 2. This process was not able any more to address the up-to-date development issues e.g. fast changing requirements, early releases for customers etc. Therefore, an agile approach like at LUH was intended to use for development projects. However, this approach did not consider an explicit design, which has been seen as very valuable at TUC especially for distributed development projects. This made TUC to consider an integration of the old waterfall design into a new development approach.

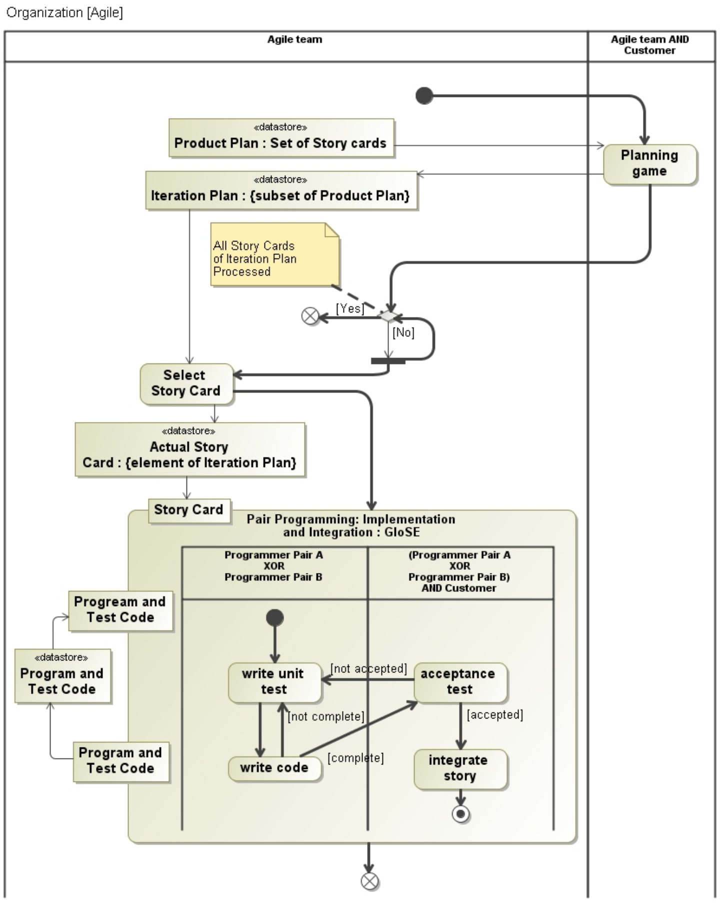
Organization [Agile]
Agile team
Agile team AND Customer
«datastore» Product Plan : Set of Story cards
«datastore» Iteration Plan : {subset of Product Plan}
Planning game
All Story Cards of Iteration Plan Processed
[Yes]
[No]
Select Story Card
«datastore» Actual Story Card : {element of Iteration Plan}
Story Card
Pair Programming: Implementation and Integration : GloSE
Programmer Pair A XOR Programmer Pair B
(Programmer Pair A XOR Programmer Pair B) AND Customer
Progream and Test Code
«datastore» Program and Test Code
Program and Test Code
write unit test
[not accepted]
acceptance test
[not complete]
[accepted]
write code
[complete]
integrate story

**Fig. 1.** Co-located Agile Development Process

## 4 The Collaborative Process Approach

The concept of collaborative processes is based on the fact that any organizations willing to collaborate need to think about how to define a collaborative process. Our concept adequately addresses this issue by using mediator patterns, which is a kind of interface, to connect two or more existing processes to a new collaborative process without changing the original ones. The mediator pattern concept is discussed in detail in [11], [12], [13] using different scenarios each incorporating an appropriate mediator pattern. A scenario is an environment that applies and illustrates the use of a mediator through example processes. Our approach provides five collaborative scenarios which allow for describing any collaboration process. Each scenario comes with an appropriate mediator pattern that makes critical collaborative aspect explicit, e.g. interface descriptions or task allocations. Thereby, two different levels of patterns are distinguished:

- Collaborating organizations have semantically different processes defined. In this case our approach provides various mediator patterns to connect these processes by explicitly including activities that address e.g. interface issues. These patterns are: *Horizontal Integration, Additive Vertical Integration, Alternative Vertical Integration, and Joint Integration*.
- Collaborating organizations have equivalent semantically processes defined. In this case the approach provides three role patterns ("AND", "OR", "XOR") to explicit allocated task responsibilities.

We will use the scenario *Additive Vertical Integration* to illustrate how a mediator pattern operates [12]. Generally, this scenario allows for parallelization of activities/sub-processes and connecting entire action chains. The appropriate mediator for this scenario is illustrated in Fig. 2. <*Action X*> is the connector from the Master's development process, which is followed by *Decomposition*. This *Decomposition* is a newly defined action that is necessary to allocate features to either <*Sub-workflow A*> or <*Sub-workflow B*>. After decomposition there exist two artifacts 2 and 5, which are created and defined in a type that they are an usable input for <*Sub-workflow A*> and <*Sub-workflow B*>.

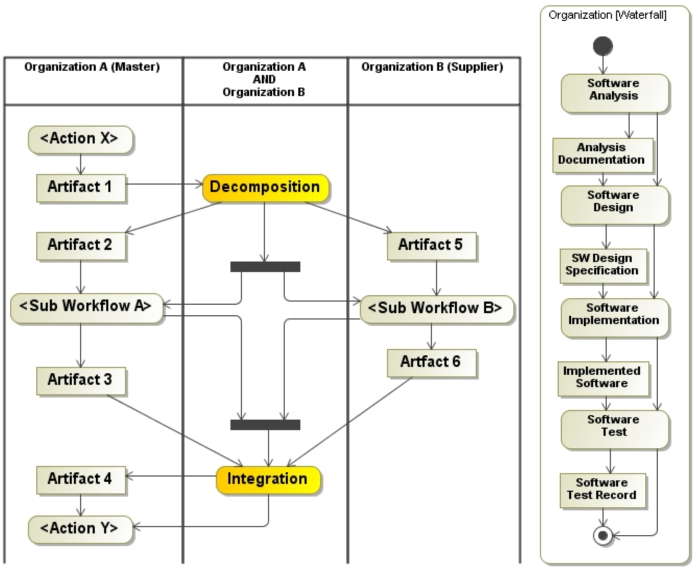

**Fig. 2.** Mediator Pattern for Additive Vertical Integration (left) and initial Waterfall process

The output artifacts 3 and 6 of these sub-workflows are of the same type as the inputs. After implementation an *Integration* of (in this case) products from <*Sub-workflow A*> or <*Sub-workflow B*> is essential to combine several features or sub-features that have been developed in different domains to one system or sub-system. As illustrated, the *Integration* is also newly defined and included as connection point. After *Integration* is done the process flows back to the Master, which is symbolized with <*Action Y*>.

## 5 Example: Deriving a Distributed Agile Development Process

For illustration, we show how we applied our approach to a student's project (11 participants at LUH and 4 participants at TUC). We formed two teams: a local team with 7 students from LUH that followed the original co-located process and a distributed team with 4 students from each site (two programmer-pairs per site) that followed the derived process for distributed XP. There was no barrier due to languages, time zones and resulting communication problems, because LUH and TUC are both German universities. The project conducts 14 development runs with duration of 4 hours each. We had 5 days including two development runs each and 4 days with one development run. The whole development project was based on XP. Although the project lasted a little over two weeks (14 working days) only, all XP practices were applied, e.g. Planning Game, Refactoring, Pair Programming, Onsite Customer, Continuous Integration etc.

Besides the XP practices discussed in Kircher et al. [10] special attention needs to be paid on the design activity which has been done implicitly in the co-located agile scenario (Fig. 1). This implicit approach is acceptable since communication paths are very short and decisions can be taken in an informal way without having anybody of the agile team ignored. In a distributed development environment design discussions are even more important due to the distributed project character - even a simple design [3] needs to be known at all sites. This was the main intention of TUC to keep explicit design activities even though an agile approach is followed. As seen in section 3 programmer pairs continuously take story cards to implement. Design discussions and definitions have to be now explicitly defined and conducted in parallel to the implementation of story cards. This is necessary because various features of story cards might affect the system design. The design adoptions in a distributed project environment need to follow a traceable approach in order to get commitment from the entire team.

### 5.1 Applied Mediator Pattern in Example: Additive Vertical Integration

In order to address the need to have development activities explicitly parallelized the *Additive Vertical Integration* scenario is predestinated to be applied. Fig. 3 illustrates the collaborative scenario. Following the mediator pattern for *Additive Vertical Integration* (Fig. 2) we have to identify <*Action X*> first, that is the starting point for integration.

Having chosen the action “Select Story” a newly defined action “Decomposition” has to be added right after a story card is selected. In this action the agile teams decide on the dedicated developer pairs that start discussions about design. A pair itself is not distributed over two locations, since this would make the XP practices even harder to fulfill by having too much communication effort. For this reason a XOR role model definition is applied in the “Pair Programming” activity, which minimizes communication overhead.

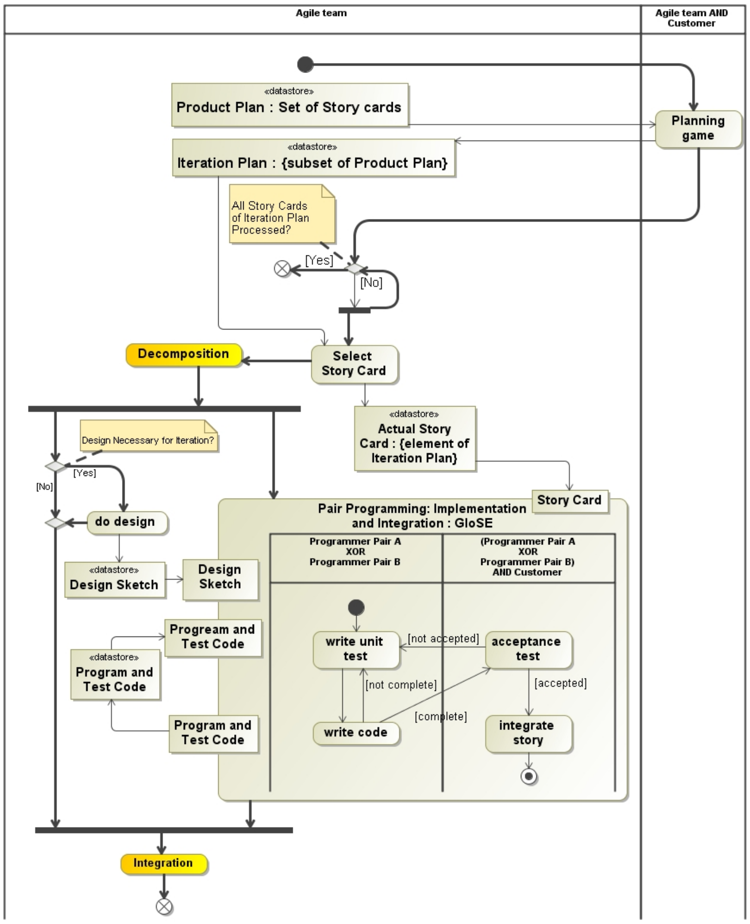
Agile team
Agile team AND Customer
«datastore» Product Plan : Set of Story cards
«datastore» Iteration Plan : {subset of Product Plan}
Planning game
All Story Cards of Iteration Plan Processed?
[Yes]
[No]
Decomposition
Select Story Card
«datastore» Actual Story Card : {element of Iteration Plan}
Design Necessary for Iteration?
[Yes]
[No]
do design
«datastore» Design Sketch
Design Sketch
Program and Test Code
«datastore» Program and Test Code
Program and Test Code
Story Card
Pair Programming: Implementation and Integration : GloSE
Programmer Pair A XOR Programmer Pair B
(Programmer Pair A XOR Programmer Pair B) AND Customer
write unit test
[not accepted]
acceptance test
[not complete]
[accepted]
[complete]
write code
integrate story
Integration

**Fig. 3.** Distributed Collaborative Process

Going further in Fig. 3 the fork node starts parallelizing the process. The left path covers the design work, which typically leads to a design sketch on a whiteboard and documents the most important design decisions for the current iteration. The design sketch will be removed, if it is no longer useful. The right path leads directly to activity *Pair Programming: Implementation and Integration*. As depicted in Fig. 3, a useful design sketch is input for this activity, which describes the implementation process itself. After implementation the story card's code is integrated into the existing software product. Following the scenario in Fig. 3 the *Join Node* terminates the parallelism. At this point another overall integration needs to take place which also includes the integration of various design approaches created on the whiteboard. This is represented by an additional *Integration* action at the end.

## 5.2 Assessment of the Derived Process

Now that we have derived a process for distributed development, we need to assess, whether it works as intended. In comparable co-located projects we observed some advantages of our co-located XP process that we wanted to include in our distributed development process:

- High developer commitment because of shared responsibility.
- Low Risk of Failure and low amount of rework because of good customer interaction. Developers

We need to investigate, to what extend the advantages from co-located development could be retained. In this example, we focus on the Risk of Failure, the amount of rework, and the truck factor.

### 5.2.1 Question 1: Commitment of Developers

**Question 1:** Does the derived distributed process explicitly allocate responsibilities in a way that leads to comparable commitment of developers as in the co-located case?

**Metrics:** We measure the additional time, students invest into the course (M1: overtime [h]) as an indicator they are highly motivated by the project. For the same reason, we count, how often students are late (M2: occurrences of being late).

**Hypothesis 1:** There is no difference in metrics M1-2 between co-located and distributed teams.

**Measurement:** During the project, tutors and observers logged all peculiarities, especially if (M1) students stayed for longer discussions or did some work (e.g. reading tutorials) at home and (M2) if a student was late during the block course.

**Findings:** Students from the global team were very interested into process issues. At three occasions they stayed more than half an hour longer, discussing about XP concepts and general software design. Such occasions were not observed with students from the local team, but we found slightly more volunteers from the local team, when we searched for volunteers to create a market-ready version of the software after class. Therefore, we rate M1 to be indifferent or even in favor of the global team. On the first few days, distributed stand-up meetings were delayed, because some developers were late (5-15 minutes). Punctuality improved during the distributed project. In the co-located project, some of the developers were regularly late, so the co-local team performed worse with respect to M2. Similar projects in past terms show that the distributed project is more typical than the co-local, here.

### 5.2.2 Question 2: Risk of Failure

**Question 2:** Does the continuous integration and the handover of new functionality to the rest of the team work as good in reducing the risk of failure as in the co-located process?

**Metrics:** New functionality should be added at a stable pace. This reduces the risk of not finishing in time with the most important requirements implemented. A stable pace is only possible, if new functionality can be integrated without major problems and reuses functionalities of existing increments. Therefore, we measure the variation of the implementation progress (M3: variation of velocity).

**Hypothesis 2:** There is no difference in the variation of the implementation progress (M3).

**Findings:** Fig. 4 shows the implementation progress of the distributed team in comparison to two co-local XP projects. The Velocity shows how much of the estimated effort was implemented each development run. The Burndown shows, how much of the initial amount of estimated work was left each day. This amount is reduced by the estimated effort of a story card, whenever it is integrated after acceptance. The Burndown should fall continuously, whereas the Velocity should ideally be constant.

If a project goes well, the velocity is constant or even growing (b in Fig. 4). If problems occur, the velocity drops (c in Fig. 4). Compared with these two co-local projects, our distributed project seems to be fine (a in Fig. 4). Remark: For b) and c) only limited data has been available which makes statistical comparisons difficult. Nevertheless, the major trend of burndown and velocity is made clear.

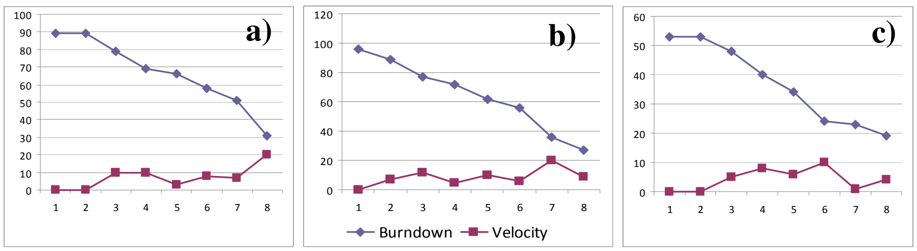

**Fig. 4.** Velocity and Burndown in a) the distributed XP project (x-axis: development runs (4h length), y-axis: estimated effort), b) a good co-local XP project, and c) a bad co-local XP project (x-axis: 8h day)

### 5.2.3 Question 3: Amount of Rework

**Question 3:** Does the distributed process support the development of a shared understanding of the design?

**Metrics:** The amount or rework was measured in two ways. Firstly, the number of story cards with bugs (“bug cards”) in relation to all story cards (M4). Secondly, the time spent on these bugs (M5).

**Hypothesis 3:** Basili and Boehm stated in [5] that the typical amount of rework is about 40-50% of the overall project effort. Our projects should be better.

**Findings:** Number of Bug Cards show a relative high value for M4 of about 37% of bugs. However, none of these bugs took long time (M5): the amount of rework is determined at about 15%. Again, these values are comparable to our co-located projects.

We investigated the communication effort of the project in detail in [16], which is about twice as high (or worse, if preparation and setup time is considered) as in local projects. As discussed in [2], strategic considerations can make this additional effort acceptable. In our case, we stay more agile, are able to react faster, and can still distribute the development between two sites.

## 6 Conclusion

More and more software is developed in distributed teams. Especially for small projects, the ability to systematically derive a process for distributed development from a well understood co-local development process is crucial. In this paper we present such an approach that allows deriving a distributed development process. We have introduced the collaborative scenario *Additive Vertical Integration* which incorporates important aspects for distributed agile development. Together with the corresponding mediator methodology this pattern helps process engineers to setup their collaborative and distributed scenario. Our approach is complemented by empirical techniques to derive the benefits and drawbacks associated with the original co-local process. This is a prerequisite for assessing, whether these properties are represented in the derived distributed approach. We give an example on how our approach (usage of Additive Vertical Integration) can be applied to a students' XP project. Since this paper provides only one quantified example (data point) concerning the effects of the use of the *Additive Vertical Integration* we cannot drive the conclusion that each and every software development project using *Additive Vertical Integration* is successful in the same way we are.

In our example, we focus on the properties developer commitment, risk of failure, amount of rework, and communication effort. Basically, we were able to derive a process that performs as good as the original co-located process at the cost of twice as high communication costs. The reasons for that are manifold; however, the use of an explicit design activity might essentially contribute to projects success since design activities are typically the foundation of any software product. The results provided give process managers one example of the potential successfulness of our collaborative scenario supporting the understanding of dependencies. Others may find our metrics and data points useful, when evaluating distributed projects.

Basically, we have shown that it is possible to derive a good distributed process with our approach. The communication intensive XP is a hard benchmark. Based on this success, we plan to apply our approach in industry strength projects. Currently, project managers are limited to ad-hoc adjustments for making a process fit for distributed development. It will be interesting to see whether our approach leads to better results.

## References

[1] Avritzer, Hasling, Paulish: Process Investigation for Global Studio Project Version 3.0. In: Second IEEE International Conference on Global Software Engineering. IEEE Computer Society (2007) ISBN 0- 7695-2920-8

[2] Bartelt, C., et al.: Orchestration of Global Software Engineering Projects (Position Paper). In: Proceedings of the 3rd International Workshop on Tool Support Development and Management in Distributed Software Projects, collocated with the 4th ICGSE 2009, Limerick, Ireland, July 13-16 (2009)

[3] Beck, K.: Extreme Programming Explained. Addison-Wesley (2000)

[4] Bird, C., et al.: Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista. In: 31st International Conference on Software Engineering, Vancouver, Canada (2009)

[5] Boehm, B., Basili, V.R.: Industrial Metrics Top 10 List. IEEE Software, 84–85 (September 1987)

[6] Braithwaite, K., Joyce, T.: XP Expanded: Distributed Extreme Programming. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 180–188. Springer, Heidelberg (2005)

[7] Bryant, S.: Double trouble: Mixing quantitative and qualitative methods in the study of extreme programmers. In: IEEE Symposium on Visual Languages and Human-Centric Computing Rome, Italy, September 26-29 (2004)

[8] Cockburn, A.: Agile Software Development. Addison-Wesley (2002) ISBN 978020169969

[9] Herbsleb, J.D., Mockus, A.: An Empirical Study of Speed and Communication in Globally Distributed Software Development. IEEE Transactions on Software Engineering 29(6), 481–494 (2003)

[10] Kircher, M., Jain, P., Corsaro, A., Levine, D.: Distributed eXtreme Programming. In: Second International Conference on eXtreme Programming and Agile Processes in Software Engineering, pp. 66–71 (2001)

[11] Klein, H., Rausch, A., Fischer, E.: Towards Process-Based Collaboration in Global Software Engineering. In: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, pp. 263–266 (2009)

[12] Klein, H., Rausch, A., Fischer, E.: Collaboration in Global Software Engineering Based on Process Description Integration. In: Luo, Y. (ed.) CDVE 2009. LNCS, vol. 5738, pp. 1–8. Springer, Heidelberg (2009)

[13] Klein, H., Rausch, A., Künzle, M., Fischer, E.: Application of Collaborative Scenarios in a Process-Based Industrial Environment. In: 36th EUROMICRO Conference on Software Engineering and Advanced Applications, Lille, France, September 01-03, pp. 327–330 (2010) ISBN 978-0-7695-4170-9

[14] Layman, L., et al.: Essential communication practices for Extreme Programming in Global Software Development Teams. Information and Software Technology, Special Issue Section: Distributed Software Development 48(9), 781–794 (2006)

[15] Meyer B.: Object-oriented software construction. Prentice Hall PTR (1997) ISBN 978-0-136-29155-8

[16] Meyer, S., Knauss, E., Schneider, K.: Distributing a Lean Organization: Maintaining Communication While Staying Agile. In: Abrahamsson, P., Oza, N. (eds.) LESS 2010. LNBIP, vol. 65, pp. 99–103. Springer, Heidelberg (2010)

[17] Prikladnicki, et al.: Distributed Software Development: Practices and Challenges in Different Business Strategies of Offshoring and Onshoring. In: Second IEEE International Conference on Global Software Engineering, pp. 262–271. IEEE Computer Society (2007) ISBN 0-7695-2920-8

[18] Ramesh, B., Cao, L., Mohan, K., Xu, P.: Can distributed software development be agile? Communications of the ACM (2006)

[19] Sangwan, Mullick, Paulish, Kazmeier: Global Software Development Handbook. Auerbach Publications, Taylor & Frances Group (2007) ISBN 0-8493-9384-1

[20] Schrage, M.: Shared Minds – The New Technologies of Collaboration. Random House, New York (1990) ISBN 0-394-56587-8

[21] Stapel, K., Lübke, D., Knauss, E.: Best Practices in eXtreme Programming Course Design. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), pp. 769–776. ACM Press (2008)

[22] Klaus-Peter, W.: Offshore Zusammenarbeit erfolgreich etabliert: Ein Praxisbericht über ein Migrationsprojekt im Maschinenbau, SIGS DATACOM Gmbh, ObjektSpektrum Nr.3, pp. 50–55 (Mai/Juni 2008)

[23] Wolf, T., et al.: Predicting Build Failures using Social Network Analysis on Developer Communication. In: 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada (2009)

# Improving Open Source Software Process Quality Based on Defect Data Mining

Wikan Sunindyo[1], Thomas Moser[1], Dietmar Winkler[1], and Deepak Dhungana[2]

[1] Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems
Vienna University of Technology, Institute of Software Technology and Interactive Systems
Favoritenstrasse 9-11/188 Vienna, Austria
{Wikan.Sunindyo,Thomas.Moser,Dietmar.Winkler}@tuwien.ac.at
[2] Siemens AG Österreich
Vienna, Austria
deepak.dhungana@siemens.com

**Abstract.** Open Source Software (OSS) project managers often need to observe project key indicators, e.g., how much efforts are needed to finish certain tasks, to assess and improve project and product quality, e.g., by analyzing defect data from OSS project developer activities. Previous work was based on analyzing defect data of OSS projects by using correlation analysis approach for defect prediction on a combination of product and process metrics. However, this correlation analysis is focusing on the relationship between two variables without exploring the characterization of that relationship. We propose an observation framework that explores the relationship of OSS defect metrics by using data mining approach (heuristics mining algorithm). Major results show that our framework can support OSS project managers in observing project key indicators, e.g., by checking conformance between the designed and actual process models.

**Keywords:** Open Source Software, Process Quality, Data Mining.

## 1 Introduction

The Open Source Software (OSS) development project uses different data management techniques for managing heterogeneous data from different data sources, e.g., Source Code Management (SCM), developer's mailing list, and bug repository, which produces a new insight on the software development [3]. A set of different data management approaches, such as data warehousing [4] or data mining techniques [5], have successfully been applied to observe the processes of OSS projects and to improve the quality of products and processes of these OSS projects [4]. Data warehousing and data mining techniques enable the observation of OSS projects processes based on measuring differences of expected requirements and the actual implementation. Measurement results can be used for improving the techniques itself and underlying methods for developing OSS.

OSS project managers need to collect, integrate and analyze heterogeneous data originating from different tools used in the OSS project, such as source code management, developer's mailing list, and bug reporting tools to observe the processes and determine the status of OSS projects [3]. Typically, project management in OSS projects is based on dynamically changing conventions between developers or contributors - usually it is performed either by senior contributors or the project initiator - while project management in conventional software project is determined prior to the actual project life time.

Observations of OSS processes are needed as an initial way to improve the quality of OSS processes and products. By observing the OSS processes, we can measure the current state of certain processes, for example the times taken to report and resolve some issues, and then find out how to reduce idle or non-productive time windows or to address bottlenecks. In OSS projects, project managers can not directly inspect the software development, since developers usually work geographically distributed including interactions through communication tools, such as SVN, mailing list, or Bugzilla [4]. Another aspect of OSS project observation focuses on the structure and culture of OSS models. The structure of OSS projects typically is more democratic compared to conventional software development projects and consists of more flexible work structures as well as global or multi-cultural communities [12]. A promising approach of project managers to observe OSS engineering processes is by analyzing data generated during the development phases, e.g., bug data, developers' communication data, and source code management data. However, the analysis process itself is not straightforward. Some preparatory steps are required to get the data ready for analysis, for example data collection, data integration, and data validation. Since the data typically originates from more than one and often heterogeneous tool, project managers need to identify the relationships between heterogeneous data sources to get meaningful patterns out of the collected data for further decisions regarding the OSS project.

Status determination with respect to the project timeline and prediction of project survivability (based on "health indicators" [18]) of OSS projects is a key activity of project managers [17, 18]. By knowing the status of OSS projects early in the development project and phases, the project managers and other stakeholders, e.g., project hosts and project contributors, can decide whether the project is healthy, sustainable and worth-supported. Among key indicators to determine the OSS project health status is a proportional number of code contributions per developer email and the number of bug status report per developer email. In "unhealthy" OSS projects, the ratio between code contributions per developer email and bug status report per developer email is imbalanced [17].

Major challenges in observing OSS processes are (1) no formal definition of engineering process design in OSS projects [8], (2) different developers work in distributed systems make it difficult to observe their work in one location, (3) heterogeneous tools and data formats in OSS projects, e.g., source code management, developers mailing list, and bug reporting systems, which hinder project managers to identify the right data to support their observation goals, e.g., to monitor the status of OSS projects [4].

Bugs resolution process is one of project observation sources. Bugzilla[1] – a web-based general purpose bug tracking and testing tool – is widely used in the OSS community. Bugzilla is used by the Mozilla project and licensed under the Mozilla Public License. Currently Bugzilla is used for tracking bugs in many OSS projects such as in Red Hat[2]. However, the use of advanced approaches, such as data mining approaches, to analyze bug report data, has not yet been intensively researched and therefore requires further investigations regarding its usefulness for OSS projects observation and quality improvement.

In this paper we propose a framework for effective engineering process observation in OSS projects. We use bug history data from Red Hat Enterprise Linux[3] (RHEL) projects as a use case for our observation framework application and use the Heuristics Mining algorithm [19] of the Process Mining tool ProM[4]. Major result is that the framework can support engineering process observation more effectively than manual approach. The analysis results on conformance checking of process models from RHEL bug history data can be used to improve the process quality, e.g., by observing that more bugs remain closed and not reopened within different RHEL versions.

The remainder of this paper is structured as follows: Section 2 discusses related work on the OSS projects domain and on data mining approaches. Section 3 motivates the research issues. Section 4 describes the solution approach of observation framework to OSS projects. Section 5 presents the results of the initial evaluation. Section 6 discusses the research results, while finally section 7 concludes the paper and presents further work.

## 2 Related Work

This section summarizes related work on some important approaches to improve the OSS process quality, especially by using process and product data from OSS artifacts. We also include some works on analyzing and mining those data to support OSS project quality improvement.

### 2.1 Observation and Improvement in OSS Projects

Some researches have been done to observe engineering processes in OSS project context in order to improve the quality of the projects. Sharma *et al.* [12] focused on creating a framework to generalize the characteristics of OSS in hybrid communities. Hybrid communities are communities of software developers who are using OSS approaches to develop software in proprietary software development projects. These developers are usually employees of a company rather than being voluntarily involved in the projects. Sharma *et al.* state that OSS development models are considered to be successful if they meet the original developers' or candidate users' software requirements

[1] http://www.bugzilla.org/
[2] http://www.redhat.com
[3] http://www.redhat.com/rhel/
[4] http://www.processmining.org/

regarding software quality, e.g., product features that should be available in final OSS products. In general, the models used for proprietary software can be based on the models of OSS projects, for example regarding the structure, culture or process to develop software, even though some of these OSS projects do not finish successfully. The OSS approaches in developing software, especially development iterations based on feedback from the developers can improve the product in order to contain fewer bugs. The framework for creating and maintaining hybrid-open source communities typically consists of structures, processes, and cultures. This means that an attempt to improve the quality of OSS projects can be adapted to other proprietary software projects as well.

Rigat [10] analyzed defect data in proprietary software development processes by using data mining approaches. He proposed to analyze the software project data to identify the underlying software development processes, e.g., why some process steps are skipped in most of the cases of handling defects. In his approach, Rigat collected development data from proprietary software projects and analyzed it by using the Heuristic Mining algorithm. The results show that there was a positive tendency of skipping the "analysis" task while the project reaches the testing phase. The reason was the developers seem not to be really considering the analysis phase in the software development (because it seems too trivial and can be changed later) and rather go directly on to the next process steps. This approach is appropriate for conventional software development projects, where the project data can be easily obtained from the project developers, but is not easily transferable to the OSS projects domain where the data typically is both geographically distributed and often only available in heterogeneous data formats or contents. Another difference between OSS and closed software development projects is that in the closed software development projects typically exists more structured project management allowing for easier project monitoring, while in OSS development projects the developers act voluntarily and the project managers need more effort to measure the activities of these OSS developers.

Mockus *et al.* performed an analysis on two OSS projects, namely the Apache Web Server and the Mozilla browser [7, 8]. Their goal was to quantify the aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution for OSS projects. They collected and analyzed the data from different sources, for example developers' mailing lists, concurrent version systems (CVS) and problem reporting databases (BUGDB). However, they focused on the use of non-integrated historical artifacts originating from different data sources to measure the quality of software. Their findings show that the Apache and Mozilla OSS development projects have core developers who control the code base. This core is no larger than 10 to 15 people, and is responsible for approximately 80% or more of newly implemented features. In this paper, we integrate both historical and current artifacts originating from different heterogeneous project data sources to get more information (e.g., current project status, times needed for bug fixing, etc.) for decision making.

Biffl *et al.* [3] proposed a project monitoring cockpit for project managers to analyze the key indicators of the OSS projects, and to investigate whether these OSS projects are healthy and sustainable or "sick" and need more treatment and support from the developers. This project monitoring cockpit used data from different data sources, e.g., source code versions, mailing list entries, and bug reports. From this project monitoring cockpit, we can expect an improvement of the OSS project indicators based on the observation results, e.g., by resolving more bugs in the same time window. However, this improvement effect cannot be derived so far without going deeper to the details of bug states themselves. Hence, research works on OSS engineering process observation are still very challenging to reveal the health status of OSS project immediately and support information for OSS project manager's decision.

The works to formulate the process model of software development, for example proposed by Mi and Scacchi [6] by using a meta-model for formulating knowledge-based models of software development. This meta-model contains a minimal set of entities necessary to describe a process, which consists of hierarchy of processes, which are composed of tasks (sets of related actions) and atomic actions.

The hierarchy may be divided into some lower parts, e.g., sub-processes, subtasks, and so forth, to achieve an arbitrary degree of decomposition. Each activity is defined in terms of process entities, namely agents that participate in the process activity, tools used by those agents in the performance of the activity, and resource that are the product of and are consumed by performance of the activity.

This meta-model can be a general model for other software development process, including the OSS development process. However, some adjustments need to be done to adapt the different process data in the OSS project, e.g., the development process data in the bug reporting systems.

## 2.2 Bug Reporting System

A bug reporting system is a very useful system that makes process observation and improvement easier. It is a software application that is designed to help quality assurance personnel and developers keeping track of reported software bugs in their work. It may be regarded as a type of issue tracking system.

Bugzilla is a "defect tracking system" or "bug-tracking system" from Mozilla.org. Defect tracking systems allow individual developers or groups of developers to effectively keep track of open issues in their product. By using Bugzilla, developers can track bugs and code changes, communicate with teammates, submit and review patches, and manage quality assurance (QA) steps [13].

Figure 1 shows the Bugzilla Life Cycle as the way the Bugzilla users check in and change the status of bugs in the Bugzilla database for software projects in general. We consider this life cycle as an expected process model of software process development, especially for OSS projects, because this life cycle can show us the process steps of changing bug status which are done by OSS developers. These actions to change the bug status we consider as engineering processes like in traditional waterfall model.

At first, a bug is introduced by developers or users as an *unconfirmed* bug (1). After the bug is confirmed, its status is changed into *new* (2). A new bug can be *assigned* (3) to other developers or *resolved* (4) directly. A resolved bug should be *verified* (5) before it is *closed* (7), but sometimes a closed bug is *reopened* (6), when new issues is revealed. Other paths could be taken, for example from a *closed* (7) bug to *unconfirmed* (1) bug, or *assigned* (3) a bug after *reopened* (6). A *verified* (5) or *resolved* (4) bug can be *unconfirmed* (1) again. These states are set by the Bugzilla developer to guide the software developers to name their bug states. However, the software developers can create their own state names based on agreement between them.

Ahmed and Gokhale [1] proposed a life cycle and resolution analysis on the bug history data for Linux kernel. In their work, they used the states from Bugzilla life cycle to examine some insights on the Linux kernel, i.e., why and how the bugs originate, contribution of the different modules to the bugs, their distribution across severities, the different ways in which the bug may be resolved, and the impact of bug severity on the resolution time.

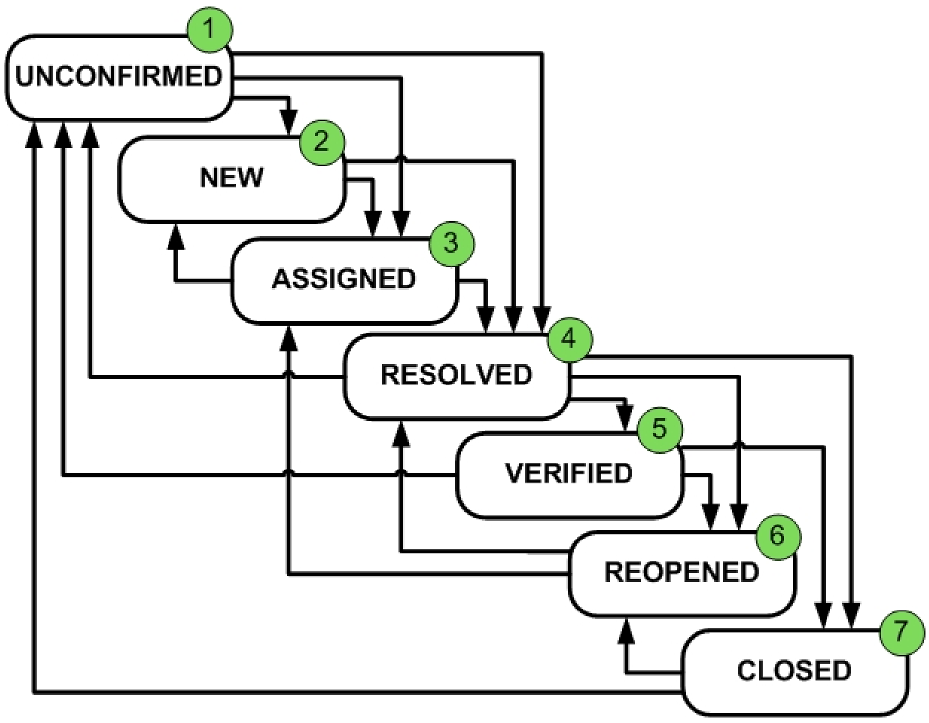

**Fig. 1.** Bugzilla Life Cycle[5]

To discover the insights, they made some statistical data analysis on cross-kernel analysis, module-level analysis, bug resolution method, and bug resolution time vs. severity. The insights actually are very useful to evaluate the status of OSS project development and can be applied to other OSS projects, not only limited to operating system application.

[5] http://www.bugzilla.org/docs/3.0/html/lifecycle.html

### 2.3 Data Mining

Data mining can be defined as the exploration and analysis, by automated and semi-automated means, of large quantities of data in order to discover meaningful patterns or rules [2]. In the context of marketing, the goal of data mining is to allow companies for improving their marketing, sales, and customer support operations through better understanding of their customers. The tasks well-suited for data mining are including classification, estimation, prediction, affinity grouping, clustering and description [2].

Data mining tasks can be classified into two tasks: namely hypothesis testing and knowledge discovery. Hypothesis testing is a top-down approach; a database recording past behavior is used to verify or disprove preconceived notions, ideas, and hunches concerning relationships in the data. Knowledge discovery is a bottom-up approach; no prior assumptions are made and the data is allowed to speak for itself. There are two kinds of knowledge discovery approaches, namely directed knowledge discovery and undirected knowledge discovery. Directed knowledge discovery attempts to explain or categorize some particular data field such as income or response. Undirected knowledge discovery attempts to identify patterns or similarities among groups of records without the use of a particular target field or collection of predefined classes [2].

In this paper, we use a hypothesis testing approach to verify the designed process model with the actual data from the bug history data of OSS projects. The results of this hypothesis testing are used to observe improvement of project quality, e.g., more bugs are closed and not reopened within one OSS project.

The data mining approach, which is applied to the process data, is called process mining. Process mining is defined as a method for distilling a structured process description from a set of real executions [15]. Similar to data mining, process mining can be used to discover meaningful patterns and rules in the process data represented as activities that stored in an event log. The goal of process mining is to extract information about processes from transaction logs [14]. Process mining assumes that it is possible to record events such that (1) each event refers to an activity (i.e., a well-defined step in the process), (2) each event refers to a case (i.e., a process instance), (3) each event can have a performer or an originator (a person or a machine that execute or initiate the activity), (4) events have timestamps and are totally ordered [16].

In [19], van der Aalst *et. al.* explain three different perspectives of process mining: namely the process perspective, the organizational perspective, and the case perspective. The process perspective focuses on the control-flow, i.e., the ordering of activities. The goal of this perspective is to find a good characterization of all possible paths, expressed in models such as Petri Nets [9]. The organizational perspective focuses on the originator field, i.e., which actors are involved in performing the activities and how they are related. The goal is to structure the organization by classifying people of systems in terms of roles and organizational units or to show relations between individual actors (i.e., by building a social network [11]. The case perspective focuses on properties of cases. Cases here can be defined as an instantiation of processes that are characterized by their path in the process or based on their originators. Cases can also be characterized by the values of the corresponding data elements.

Heuristics mining is an experience-based process mining technique that can help in problem solving, learning and discovery. Right now, the heuristic mining algorithm is implemented as HeuristicsMiner plug-in in the process mining (ProM) tool [19]. HeuristicsMiner is a practical applicable mining algorithm that can deal with noisy data. It can be used to express the main behavior (i.e., not all details and exceptions) registered in an event log. The HeuristicsMiner algorithm consists of the following three steps: (1) construct the dependency graph; (2) for each activity, construct the input- and output expression; and (3) search for long distance dependency relations [19]. In this paper, we use HeuristicsMiner algorithm to analyze the bug history data from different OSS project versions to compare the frequency of using bug states in those OSS project versions.

## 3 Research Issues

The OSS project managers need to be able to know the status of OSS project developments [17], so they can make further decisions on the projects, e.g., to assign some experts to handle difficult tasks, to add some new features to address some bug issues. One way to support this requirement is by giving ability to the OSS project managers to observe the engineering processes of OSS projects.

However, observing engineering process of OSS projects is a difficult task, because (1) there is no formal engineering process design in OSS project [8], (2) the developers work in distributed systems [8], (3) different tools are used in developing the software, e.g., source code management, developers mailing list, bug reporting systems [4]. From these challenges, we derive two research issues to provide a foundation for OSS engineering process observation.

**RI-1: How can the project managers observe the engineering processes effectively?** An effective engineering process observation involves appropriate data source selection and the using of automated data collection to make the observation faster and produce more analysis results.

One possible source to get engineering processes from OSS projects is a bug reporting system. The advantage of bug reporting systems to other sources is that in the bug reporting system, we can find a bug life cycle that we can consider as an engineering process model. To be able to observe the engineering processes in OSS projects effectively, the project managers should check the conformance of designed process model with the process model from actual data. We propose a framework for collecting and analyzing data from bug reporting systems for an effective engineering process observation.

Data collection and integration is one of most difficult parts in the engineering process observation framework, because we have to deal with a large number of historical data that is stored in the bug reporting systems. A manual data collection and integration is error-prone and takes a lot of time [4]. Hence, we propose an automated approach to collect and integrate bug historical data from the bug reporting systems.

**RI-2: How to validate the designed process model with the actual engineering process data?** To evaluate actual engineering process data we collected from bug reporting system, we propose to make a conformance checking analysis between the actual data and the designed engineering process model. The results of analysis can show similarities or deviations between the process model derived from actual data and the designed process model, hence support the justification of the project managers on OSS projects status. We defined two research hypothesis based on the actual data to be validated by the experiments.

***RH1. The RHEL developers are following the naming and the ordering of the bug states from Bugzilla Lifecycle.*** Different RHEL versions will have different number of states used for addressing bug. In this study we use 3 different versions of RHEL namely RHEL 4, RHEL 5 and RHEL 6. A process model from a version of RHEL will have the same number of bug states to the other versions and to the designed process model from Bugzilla life cycle. IF $\phi$ is the designed process model, and $\psi$ is a certain RHEL version (from version 4 to 6), and $P$ is a function to get the number of bug states from designed process model or from certain RHEL version process model, THEN we can formulate following null hypothesis.

$$\boldsymbol{H01}\text{: } \{\exists \psi \in (V4, V5, V6) \mid P(\psi) = P(\phi)\} \qquad \text{(eq. 1)}$$

For every RHEL version, the number of states used for addressing the bugs is the same as the number of states used in the Bugzilla life cycle. This hypothesis is created to check whether the RHEL developers are using the Bugzilla life cycle as their designed process model to develop RHEL products and address the bugs inside the products.

***RH2. The RHEL developers are using all bug states for each bug history in the same number of frequency.*** For different bug states in one RHEL version, the number of frequency should be the same. IF $s_i$ is a bug state and $s_{i+1}$ is another bug state after $s_i$, and $N$ is a function to get the frequency of bug states using in one RHEL version, THEN we can propose following null hypothesis.

$$\boldsymbol{H02}\text{: } \{\forall s_i, s_{i+1} \mid N(s_i) = N(s_{i+1})\} \qquad \text{(eq. 2)}$$

Following the designed process model, the number of frequency between the bug states used in one RHEL version could be similar or vary. In this hypothesis, we want to check whether the frequencies are all the same, and what bug states are used more often than the others. The result of this hypothesis will be useful to endorse the using of similar bug states with then designed process model rather than create new bug states.

## 4 Solution Approach

In this section we address the research issues in section 3 and identify the validity threats of our current study.

### 4.1 Effective Engineering Process Observation

To observe the engineering processes in OSS projects effectively, we propose an observation framework which consists of some steps and tools to support engineering process data collection and analysis for OSS project managers.

Figure 2 shows the framework for observing bug history data. This framework involves 4 different levels namely data source, data collection, data integration and validation, data analysis and presentation.

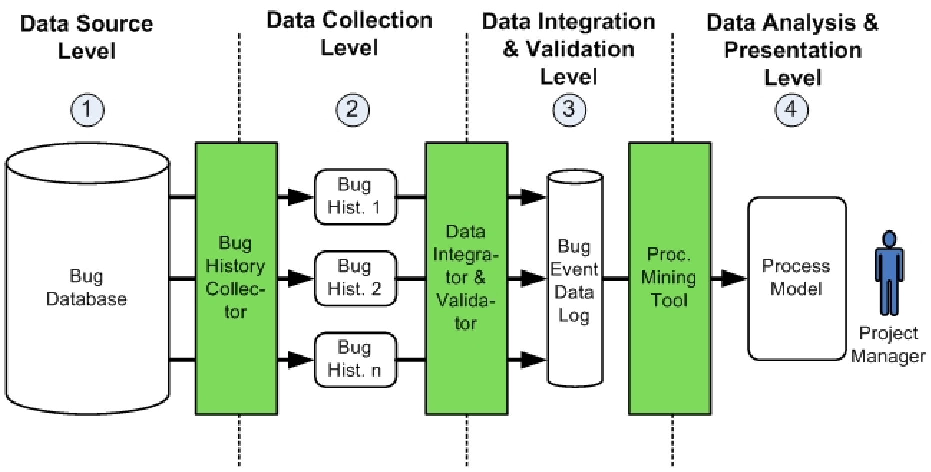

**Fig. 2.** Framework for Bug History Observation

(1) In the *data source level*, we have bug database which contains all bugs information that are used in software development. However, in this case, we don't need all of those data and focus on bug history data, which can be extracted from the bug data.

(2) In *data collection level* we extract and collect bug history data from filtered bugs. Filtering on the bug database for example focusing on project, version, status (open or closed bugs), times duration of bugs, priority, severity, or bug reporter. We collect the bug history data from the bug database by using a bug history collector.

(3) *Data integration and validation.* A bug history is a set of state transitions of one bug id. We collect bug history data from some bug ids, integrate and validate them in the bug event data log by using data integrator and validator.

(4) *Data analysis and presentation*. The even data log from previous level is analyzed by using Process Mining tool. The analysis results are presented to the project managers.

In this paper, we use Bugzilla on RHEL[6] as case of our bug database on OSS projects. RHEL is a Linux distribution produced by Red Hat Company and is targeted toward the commercial market, including mainframes. RHEL is a popular, quite stable and mature OSS development project that is well-supported by the company and community.

Currently, in the Red Hat Bugzilla browser, there are in total 21.640 bugs reported for RHEL version 4 (2.300 open bugs and 19.340 closed bugs), 41.187 bugs reported for RHEL version 5 (6.441 open bugs and 34.746 closed bugs) and 23.768 bug reported for RHEL version 6 (7.465 open bugs and 16.303 closed bugs).

[6] https://bugzilla.redhat.com

We focus on the use of closed bugs data from RHEL 4, RHEL 5, and RHEL 6. The usage of closed bugs only allows for an easier analysis of the process model based on the historical data, especially on the status changes of each bug.

The selection of using closed bugs data in our research is based on the assumption that closed bug data contains all necessary steps which are required from introducing bugs till closing the bugs in a complete cycle. Open bugs may contain a lot of introductionary bug states and unnecessary intermediate states that may hinder an effective and efficient generation of valid process models. We also have conducted preliminary experiments on previous versions of RHEL (version 2.1 and 3), but the resulting data contains more unnecessary or duplicate bug states which were getting reduced in the next versions. The usage of data from version 4, 5 and 6 is representative enough to show the trend of states reducing between versions.

To collect and integrate bug history data from the bug database automatically, we made a bug history data collector tool written in Java. This tool collects bug information from Red Hat Bug Repository, like bug id, bug severity, priority, operating system, bug status, resolution, and short description of the bug.

The access to the bug data can be done by using a web service interface that is provided by Bugzilla, namely XML RPC[7]. XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism.

XML-RPC works by sending a HTTP request to a server implementing the protocol. The client in that case is typically software wanting to call a single method of a remote system. Multiple input parameters can be passed to the remote method, one return value is returned. The parameter types allow nesting of parameters into maps and lists, thus larger structures can be transported. Therefore XML-RPC can be used to transport objects or structures both as input and output parameters.

By using the XML-RPC interface which is provided by Bugzilla, we also implemented the bug history collector to collect the history of status changing of the bugs, especially for closed bugs. We collect the history of the bugs in order to learn the processes performed to change the state of a bug from one state to another state. We focus only on closed bugs to identify the complete life cycle of bug. There always exists the possibility to reopen a prior closed bug. In this case, we just consider the latest state of the bug.

## 4.2 Evaluation of Engineering Process Model

For analyzing the bug history data, we used a process analysis tool called ProM[8]. This tool has capabilities to discover process model, make conformance checking between expected process model and the process model generated from actual data, and make performance analysis on process model for process improvement. There are a lot of plug-ins and algorithms to discover the process model from actual data. One of them is heuristics mining.

The heuristics mining is a process mining algorithm in ProM which is based on the frequency of the patterns. The most important characteristic of the heuristics mining is

[7] https://bugzilla.redhat.com/xmlrpc.cgi

[8] http://www.processmining.org

the robustness for noise and exceptions. We use the heuristics mining to analyze event log from bug history data to find out the process model from actual data, rather than designed process model.

We identified and addressed threats to internal and external validity of our evaluation results as follows.

*Threats to internal validity - Numbers of states.* As we have conducted previous experiments using fewer data, there is a tendency of increasing of the numbers of states as new data is added. So we put more focus on the frequency of states taken during development, rather than only the number of states in the process model, since the number of states can be unnecessary increasing, while the top states remain stable.

*Threat to external validity.* In this study we focus on three versions of RHEL projects with similar size and characteristics. The selection of these homogeneous OSS projects may raise concerns whether the resulting process models are also valid for other project contexts. While we assume our approach to hold for projects similar to our study objects (i.e., under Red Hat or similar managements with active and large developer community), further research work is necessary to investigate projects with strongly differing characteristics.

## 5 Results

In this section, we report the results of our study in section 4. The results show the application of our observation framework, the using of our automated data collection and integration tool, and the analysis results of our evaluation on the engineering process models.

### 5.1 Engineering Process Observation Framework

To observe software engineering processes from the bug database effectively, we applied the observation framework from figure 2. In the application, we take Bugzilla report of RHEL projects as data sources, Bug History Data Collector as an automated data collector, integrator and validator, and Process Mining (ProM) tool for data analysis and presentation. By following this framework, the observation works can be done effectively. The project managers can make conformance checking of process models from actual bug history data to the designed process model from Bugzilla life cycle.

We have implemented and used a Java-based Bug History Data Collector tool to collect, integrate, and validate bug history data from Bugzilla database. This tool can select bug ids based on the OSS projects and versions. As results, we have collected 1500 data sets from 3 RHEL versions (4, 5 and 6). These data will be analyzed for process model conformance checking with the Bugzilla life cycle.

### 5.2 Evaluation Results

The evaluation is done by analyzing the bug history data sets from three different RHEL versions (4, 5 and 6) by using Heuristics Mining algorithm from Process

Mining (ProM) tool. We analyzed the number of states in the process models generated by ProM and counted the frequency of each state for each RHEL version. We compare the results with the designed process model from Bugzilla life cycle. We evaluate the actual data by addressing two hypotheses we have defined in the third research issue in section 3.

**Table 1.** Name of States used in different RHEL versions and Bugzilla Life Cycles

| States | Bugzilla LC | RHEL 4 | RHEL 5 | RHEL 6 |
|---|---|---|---|---|
| UNCONFIRMED | ✓ | × | × | × |
| NEW | ✓ | ✓ | ✓ | ✓ |
| ASSIGNED | ✓ | ✓ | ✓ | ✓ |
| RESOLVED | ✓ | ✓ | × | × |
| VERIFIED | ✓ | ✓ | ✓ | ✓ |
| REOPENED | ✓ | ✓ | ✓ | ✓ |
| CLOSED | ✓ | ✓ | ✓ | ✓ |
| NEEDINFO | × | ✓ | ✓ | ✓ |
| MODIFIED | × | ✓ | ✓ | ✓ |
| ON_QA | × | ✓ | ✓ | ✓ |
| RELEASE_PENDING | × | ✓ | ✓ | × |
| QA_READY | × | ✓ | × | × |
| NEEDINFO_REPORTER | × | ✓ | ✓ | ✓ |
| INVESTIGATE | × | ✓ | ✓ | ✓ |
| NEEDINFO_PM | × | ✓ | × | × |
| PROD_READY | × | ✓ | × | × |
| FAILS_QA | × | ✓ | ✓ | × |
| PASSES_QA | × | ✓ | × | × |
| NEEDINFO_ENG | × | ✓ | ✓ | × |
| ASSIGN_TO_PM | × | ✓ | ✓ | ✓ |
| ON_DEV | × | ✓ | × | ✓ |
| SPEC | × | ✓ | × | × |
| POST | × | × | ✓ | ✓ |
| **# States** | **7** | **21** | **15** | **13** |

***The RHEL developers are following the naming and the ordering of the bug states from Bugzilla life cycle.*** Table 1 shows the comparison of bug states used in the Bugzilla life cycle, RHEL 4, RHEL 5, and RHEL 6. From Table 1 we can see different bug state names are used during addressing bug in different RHEL versions.

This result shows us that the developers don't really follow the process model in the Bugzilla Life Cycle. The Bugzilla Life Cycle is build to give a guidance for the developers in handling the bug issues in the project. However, in the implementation, the developers have capability to introduce and modify new bug states as long as this is mutually agreed among the developers.

The using of too many different states in the Bugzilla sometimes is confusing and makes a lot of confusion among the developers. Some intervention should be taken to make common understanding about the meaning of the states and when it should be used to prevent ambiguity and duplication of similar states. From this result, we can see how the using of bug states evolves in different RHEL versions that bring more common understanding between the developers about the using of states in handling the bug issues.

From table 1, we can see that there are differences between the names and orders of bug states from RHEL different versions and those from Bugzilla Life Cycle. The names and orders of bug states in the Bugzilla Life Cycle are focusing on main states of the bugs and minimal requirements of the bugs with assumption that the bug information and the bug states are self-explained. However, in the reality, not all bug information and bug states are understandable by other developers. Thus, some states are created to ask for further explanation, e.g., needinfo, needinfo_reporter, needinfo_PM, and needinfo_eng. Thus four new states represent a need for further information from other parties, e.g., reporter, project manager, or engineer.

Other new states also related with QA (quality assurance), e.g., on_QA, QA_ready, fails_QA, and passes_QA, which mean that the quality assurance become a part of improvement in dealing with the bug. On_QA means that the bugs resolving is still on quality assurance. QA_ready means the bugs are ready to enter the quality assurance phase. Fails_QA means that the bugs are failed in quality assurance testing. Passes_QA means that the bugs have passed the quality assurance testing. However, these QA-related states are introduced in RHEL version 4 and not continued in version 5 and 6, means that the QA-related states are not really useful in dealing with the bugs.

Other new states are including modified, investigate, release_pending, prod_ready, on_dev, spec, and post. These states are more specific to some conditions, e.g., modified means that the bugs are still modified, investigate means that the developers need more investigation on the bugs, release_pending asks for pending of the product release, prod_ready means that the product is ready, on_dev means that the product is on development, spec asks for specification, and post means that the product has been posted.

From this result, we can see that by analyzing the different states using bug historical data, we can learn how the developers using the bug states to communicate the idea how to deal with the bugs. The change of using bug state names in different versions also show the importance of the bug state names in handling the bugs, some names are remain, but some others are not used anymore.

***The RHEL developers are using all bugs states for each bug history in the same number of frequency.*** Table 2 shows the frequencies of the using of each bug state in the bug history. From Table 2, we can see that the frequencies for different bugs in one RHEL version are not similar. The usage of some states is more frequent then of other states.

By seeing this result, we can learn how the developers deal with the bug issues by using Bugzilla. The developers seem to uses some state rather than the other states. The “closed” state is on the top of each version means that (1) whatever state as starting state, all states tend to go to closed states, (2) there are some possibility to reopen the closed bug and close it again (especially in RHEL version 4). From this result, we can see the priority of the developers, in managing bug states and suggest for improvement, e.g., reducing the number of bug states to make the development/bug handling more efficient.

## 6 Discussion

In this section, we discuss our results based on the research issues.

### 6.1 Engineering Process Observation Framework

The effective engineering process observation can be done if we follow some directions, rather than we do it randomly. So as first step, we build an observation framework which consists of necessary steps and tools as directions to do engineering process observation, based on our experience on data collection and analysis.

We followed the observation framework we defined earlier to improve the engineering process in OSS projects. The benefit of this framework is an effective and systematic approach to collect and analyze data from the bug database to support the project managers' decision making to improve the process quality in OSS projects. The cost of this approach is on preparing the framework and building the automated tools to be used in applying the framework.

Data collection and integration for engineering process analysis can be done automatically by using our bug history data collector tool. This tool is specially built for collecting, integrating and validating bug history data from Bugzilla database. The benefit of this tool is that we can reduce errors, risks and efforts on manual data collection and integration. The cost is on the efforts to prepare the tool, but once we have the tool, we can collect and integrate other data faster and easier.

### 6.2 Evaluation Results

The evaluation of actual engineering process model is done by checking its conformance to the designed process model. The actual engineering process model can be generated by using Heuristic Mining algorithm, e.g., process model which is generated from RHEL 6 bug history data shown in Figure 3. This process model can be conformed to the designed process model (Bugzilla Life Cycle) shown in Figure 1 to see the similarities and differences of both process models.

From the process model generation on 3 RHEL versions, we answer the two research hypotheses as follows.

***The RHEL developers are following the naming and the ordering of the bug states from Bugzilla life cycle.*** As shown in table 1, we can see the differences of number of states used in the designed process model (Bugzilla life cycle) and states used in the generated process models from RHEL 4, RHEL and RHEL 6.Therefore $\{\exists \psi \in (V4, V5, V6) \mid P(\psi) \neq P(\phi)\}$ thus we can reject our null hypothesis ***H01.***

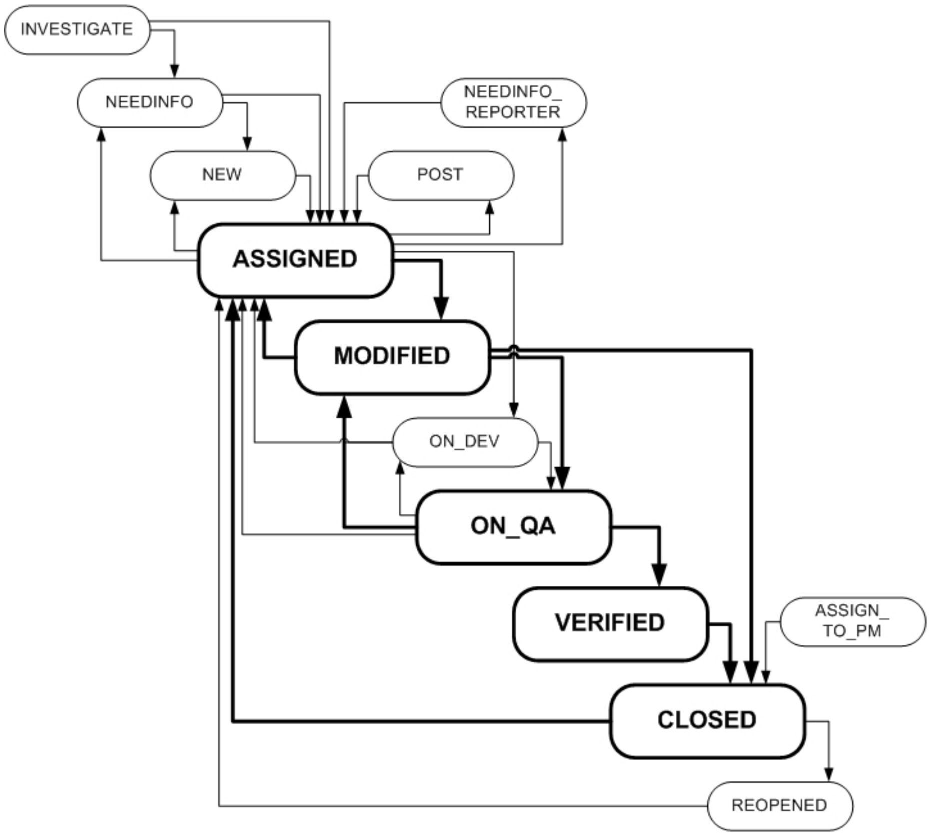

**Fig. 3.** Process Model from RHEL version 6

An interpretation of these results can be the fact that the number of bug states available and used in the different RHEL versions decreases with the version number, meaning that states which were not used at all or only very infrequently are removed for the next RHEL version.

The using of extra bug states of the RHEL versions comparing to the original Bugzilla Life Cycle represents the needs of developers to enhance their understanding on handling the bug issues. The decreasing of the bug states used between different RHEL versions means the developers have come to some convergences of understanding, such that they don't use some bug states, due to better way to explain and deal with the bugs, e.g., some bug states which means need for more information from different parties, means that the explanation about the bugs is getting better than previous version. The better explanation of the bug to other developers can increase faster bug handling, thus increasing the productivity and process quality.

***The RHEL developers are using all bug states for each bug history in the same number of frequency.*** As shown in table 2, each bug state is used in different frequency by the developers. Some bug states are used more often than the others. Therefore $\{\forall s_i, s_{i+1} \mid N(s_i) \neq N(s_{i+1})\}$ thus we can reject our null hypothesis ***H02.***

An interpretation of these results can be the fact the typically OSS projects do not follow a strict waterfall-like software engineering process, but rather a sometimes mixed dynamic software engineering process.

**Table 2.** Frequency of States for Different Versions of RHEL

| States | Version 4 | | Version 5 | | Version 6 | |
|---|---|---|---|---|---|---|
| | Occ. (abs) | Occ. (rel) | Occ. (abs) | Occ. (rel) | Occ. (abs) | Occ. (rel) |
| CLOSED | **557** | **41.0 %** | **578** | **30.3 %** | **546** | **33.6 %** |
| ASSIGNED | **282** | **20.8 %** | **407** | **21.3 %** | **255** | **15.7 %** |
| NEEDINFO | **150** | **11.1 %** | **204** | **10.7 %** | 6 | 0.37 % |
| MODIFIED | **126** | **9.3 %** | **345** | **18.1 %** | **306** | **18.9 %** |
| REOPENED | **52** | **3.8 %** | 8 | 0.4 % | 1 | 0.1 % |
| RESOLVED | 47 | 3.5 % | | | | |
| ON_QA | 29 | 2.1 % | 67 | 3.5 % | **259** | **16.0 %** |
| RELEASE_PENDING | 25 | 1.8 % | 61 | 3.2 % | | |
| QA_READY | 25 | 1.8 % | | | | |
| NEW | 16 | 1.2 % | 44 | 2.3 % | 7 | 0.4 % |
| NEEDINFO_REPORTER | 11 | 0.8 % | 14 | 0.7 % | 2 | 0.1 % |
| INVESTIGATE | 10 | 0.7 % | 2 | 0.1 % | 2 | 0.1 % |
| VERIFIED | 9 | 0.7 % | **122** | **6.4 %** | **199** | **12.3 %** |
| NEEDINFO_PM | 3 | 0.2 % | | | | |
| PROD_READY | 3 | 0.2 % | | | | |
| FAILS_QA | 3 | 0.2 % | 10 | 0.5 % | | |
| PASSES_QA | 2 | 0.1 % | | | | |
| NEEDINFO_ENG | 2 | 0.1 % | 1 | 0.1 % | | |
| ASSIGN_TO_PM | 2 | 0.1 % | 2 | 0.1 % | 1 | 0.1 % |
| ON_DEV | 2 | 0.1 % | | | 8 | 0.5 % |
| SPEC | 1 | 0.1 % | | | | |
| POST | | | 43 | 2.3 % | 31 | 1.9 % |

Some bug states are more frequent than the others, for example closed, assigned, modified, verified, and On_QA.

Closed state is on the top of all versions justifies this state as the goal of other states. All other states are tending to finish in the closed state, even though there are some options to reopen the closed bugs.

Assigned state represents the beginning of the bug state which should be assigned among the developers. When the bug issue is introduced for the first time, there is an opportunity whether to offer the bug handling to a specific person or to ask other developers publicly to voluntarily taking the chance for handling the bug issue. At some points, the bug reporter should ensure that each bug issue has been assigned to another developer such that the bug issue can be solved immediately.

Modified state represents the condition where the bug has been modified. The result shows that the numbers of modified bugs are increasing across different versions, means that more bugs are identified as modified rather than other states that are less frequent and not used in later version (e.g., resolved, QA_ready).

Verified state becomes more common across different versions. More bug issues are needed to be verified during their handling, and in other side, the resolved state becomes extinct in later version.

On_QA state is also increasing across different versions, while other states related to QA (QA_ready, Fails_QA, Passess_QA) become extinct, means the QA-related states converge to On_QA state.

Other states are used not so frequently. From this result, we can see how the developers focus on some important states rather than the others. The understanding of the developers in dealing with the bug states is also increasing the bug resolving period, hence improving the process quality.

## 7 Conclusion and Further Work

Observations of OSS processes are needed as an initial way to improve the quality of OSS processes and products. OSS project managers need to collect, integrate and analyze heterogeneous data originating from different tools used in the OSS project, such as source code management, developer's mailing list, and bug reporting tools to observe the processes and determine the status of OSS projects.

Bugs are an important source for project observation. However, the use of advanced approaches, such as data mining approaches, to analyze bug report data, has not yet been intensively researched and therefore requires further investigations regarding its usefulness for OSS projects observation and quality improvement.

In this paper, we have explained the contribution of an observation framework in improving the process quality in OSS projects, e.g., by observing the end states of bugs are not reopened frequently. We used bug history data from RHEL projects as a use case for our observation framework application and use the Heuristics Mining algorithm of the Process Mining tool ProM. The analysis results on conformance checking of process models from RHEL bug history data can be used to improve the process quality.

Future work will include the integration of additional project data sources, for example from source code management and communication artifacts, in order to perform new types of communication metrics, as well as other process metrics in observing engineering process in OSS projects.

**Acknowledgements.** This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

## References

1. Ahmed, M.F., Gokhale, S.: Linux Bugs: Life Cycle and Resolution Analysis. In: The Eighth International Conference on Quality Software (QSIC 2008), pp. 396–401 (2008)
2. Berry, M.J.A., Linoff, G.: Data Mining Techniques For Marketing, Sales, and Customer Support. John Wiley & Sons, Inc., Toronto (1997)

3. Biffl, S., Sunindyo, W., Moser, T.: A Project Monitoring Cockpit Based On Integrating Data Sources in Open Source Software Development. In: Twenty-Second International Conference on Software Engineering and Knowledge Engineering (SEKE 2010), San Fransisco Bay, USA, pp. 620–627 (2010)
4. Biffl, S., Sunindyo, W.D., Moser, T.: Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release Software Projects. In: International Conference on Complex, Intelligent and Software Intensive Systems, pp. 360–367. IEEE Computer Society (2010)
5. Gegick, M., Rotella, P., Tao, X.: Identifying security bug reports via text mining: An industrial case study. In: 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 11–20 (2010)
6. Mi, P., Scacchi, W.: A meta-model for formulating knowledge-based models of software development. Decis. Support Syst. 17, 313–330 (1996)
7. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the Apache server. In: 22nd International Conference on Software Engineering, pp. 263–272. ACM, Limerick (2000)
8. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol. 11, 309–346 (2002)
9. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491. Springer, Heidelberg (1998)
10. Rigat, J.: Data Mining Analysis of Defect Data in Software Development Process. In: Department of Technology Management Division of Information Systems, p. 65. Eindhoven University of Technology, Eindhoven (2009)
11. Scott, J.: Social Network Analysis. Sage, Newbury Park (1992)
12. Sharma, S., Sugumaran, V., Rajagopalan, B.: A framework for creating hybrid-open source software communities. Information Systems Journal 12, 7–25 (2002)
13. The Bugzilla Team: The Bugzilla Guide - 3.0.11 Release (2009)
14. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47, 237–267 (2003)
15. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16, 1128–1142 (2004)
16. van Dongen, B.F., van der Aalst, W.M.P.: A Meta Model for Process Mining Data. In: CAiSE 2005 Workshops, pp. 309–320 (2005)
17. Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., Tjoa, A.M.: Monitoring "Health" Status of Open Source Web Engineering Projects. International Journal of Web Information Systems 1(2), 116–139 (2007)
18. Wahyudin, D., Schatten, A., Mustofa, K., Biffl, S., Tjoa, A.M.: Introducing "Health" Perspective in Open Source Web-Engineering Software Projects, Based on Project Data Analysis. In: IIWAS International Conference on Information Integration, Web-Applications and Services, Yogyakarta Indonesia (2006)
19. Weijters, A.J.M.M., van der Aalst, W.M.P., de Medeiros, A.K.A.: Process Mining with the HeuristicsMiner Algorithm. BETA Working Paper Series. Eindhoven University of Technology, Eindhoven (2006)

# The Many Forms of Process Improvement – Results of an International Survey

Tom McBride[1] and Marion Lepmets[2]

[1] Faculty of Engineering and IT, University of Technology, Sydney, Australia
Tom.McBride@uts.edu.au
[2] Public Research Centre Henri Tudor, 29 J. F. Kennedy ave., Luxembourg
Marion.Lepmets@tudor.lu

**Abstract.** When discussing process improvement, different authors assign different goals of improvement and describe different methods of improvement. Several process improvement methods and theories of organizational performance are examined to reveal how each might give rise to different concepts and concerns of process improvement. There is little empirical information available to support or refute such expectations about improvements made to software development processes, whether through formal process improvement initiatives or through responses to changes in the development environment. In the absence of such information those involved in process improvement, from standards development through to consultants and those who implement process improvement projects, risk making poor decisions about what changes should be made to the processes. A project to develop and run an international survey was conducted by a number of researchers from different parts of the world to understand various forms of process improvements, their goals and motivations. The result of the study indicates that the motivations for process improvement are not matched by the improvement goals that are themselves not achieved by the implemented improvements. The study also contradicted commonly cited beliefs that process improvement changes are seldom reviewed, are seldom permanent and can make the situation worse.

**Keywords:** Process improvement, motivation, software development, service management.

## 1 Introduction

Organizations have been changing and improving their processes for many years in response to competition, a change in regulations or any number of other reasons. Although there are several methods and approaches to process improvement, we know little about what actual changes may have been made or what effect they had on process performance and the organization as a whole. Did the change achieve the improvement? There is still very little empirical data to demonstrate a relationship between process improvement initiatives and improved process performance. Process changes arising from any specific process improvement method may be implemented properly but evidence that such improvements make effective changes in process performance is difficult to find. There have been some studies of the overall effect and return on investment of the CMM™ and CMMI ™ [1-4] but few studies get down to the level of considering specific process changes and their effects. In the absence of such information, organizations risk investing in process improvement initiatives or process changes that are ineffective at best. We also wanted to discover/find out the organizational level at which process improvements are initiated and the level on which the improvements are finally implemented.

To overcome this lack of empirical information and to provide some much needed basis for reasoning about process improvements a project was initiated to gather data internationally. The overall question to be investigated was:

**How do organizations improve their processes?**

In order to attract a wide range of responses the survey deliberately did not restrict itself to improvements or changes arising from a formal process improvement initiative, i.e. improvements that follow process model or standard based process assessment.

This paper proceeds by describing different aspects of process improvement that might inform any discussion on links between process changes and the effects of those changes, and the description of the research method. The overall research question is decomposed into four specific questions; data is presented, analyzed and discussed. The limitations of this study are described before a discussion of the overall findings, followed by a summary and description of possible future work.

## 2 Improvement

For the most part, improvements in software quality have been achieved through the improved professional discipline [5] adoption of different methods of specifying requirements [6], different methods of project estimation and planning [7], better tools and languages among a number of other innovations. The general aims of process improvement are to change the attributes of an organization's products or services by changing the processes used to develop, deliver or operate them [8-11] or to change the system of processes to become faster, more flexible [12, 13]. Process improvement has various concepts, goals, methods and measurement scales. Although a process improvement expert might assume that the concepts and concerns of software process improvement are as clear to their audience as they are to them this is seldom true, leading to confusion about the applicability of the findings or advice to any given situation.

Perceptions of process affect significantly the nature of improvement. A process can be considered as the entire value chain, spanning many activities and several parts of the organization [14] for which interfaces and smooth functioning between different parts of the organization tend to be the focus of improvement initiatives. When a smaller part of the overall value chain is considered as the process, in which people collaborate to carry out a single transformation of inputs to outcomes [15], process improvement will tend to favour the goals of that process in isolation. Improvements to the larger overall process can consider improvements in relation to organizational strategic goals regardless of where those improvements might be made within the process. Improvements that are made without reference to organization's strategic goals are unlikely to contribute to the organization as much as they could. On the other hand, there is little information available about mapping organization's strategic goals to software development processes, either fine grained or coarse grained.

The motivation for process improvement differs between developers, project managers and senior managers [16]. The differences reflect the expected concerns of different organizational levels [14, 17] and can be expected to affect not only goals and support for different process improvement initiatives but also perceptions of their success. Developers favour bottom up initiatives, initiatives that affect resources (presumably favourably) and initiatives that involve top down commitment. Project managers, on the other hand, are motivated by initiatives that affect visible project success and resources while senior managers are motivated by initiatives that affect visible business success and the ability to meet organization's business performance targets. So what will be seen by one group as a worthwhile process improvement will not necessarily be seen as worthwhile by other groups.

Process improvement methods also vary from the very informal to the very formal. Agile software development practitioners seem to favour nothing more formal than a retrospective [18] or simply a general intention to learn and adapt [19]. At the other end of the scale are the formal process improvement methods of SPICE [20] or CMMI [11] that trace their origins to Total Quality Management. While the agile developers seem to assume that the developers themselves are best placed to know what should be improved in the specific circumstances [21], SPICE and CMMI seem to be more concerned with both the project managers' and senior managers' perception of improvement. One is practice based, the other process based but both seek to respond to perceived opportunities to improve the way work is done.

Obviously a process improvement should make the situation better in some way but just what "better" might be also varies. While individual improvements might favour, for example, improved social relations within a development team different scales of improvement require that each improvement contribute to specific dimensions of process performance. The most familiar scale of process performance is that of capability and maturity [22] in which process improvements are expected to move the process from chaotic performance, through managed, defined, quantitatively managed to optimizing. The first three of these capability levels are aimed at getting the process under control and able to repeat its performance reliably [5]. However, there is no assumed or embedded process goal other than the declared purpose of the process. For example, the improvements in process performance are not specifically linked or aligned to the improvement in the productivity of the process. The overall effect of improving processes on this scale is to improve project performance and quality [2, 4]. In an effort to find explanations why some manufacturers seem to improve all manufacturing capabilities instead of the expected trade-off between capabilities, Ferdows and De Meyer [23] found that the successful manufacturers improved quality, dependability, speed and cost efficiency in that sequence. Although

Flynn and Flynn [24] are critical of the lack of empirical evidence for the sand cone model, they reach a very similar conclusion about the sequence of cumulative capabilities. The sand cone model proposes that organizational capabilities build on lower level capabilities and that improvement at one level requires an even larger improvement in the level below it. In this model, quality is the lowest level capability so organizations are well advised to develop that capability first. Ferdows and de Meyer's proposed "sand cone" model is similar to the SPICE and CMMI capability scale in that each level builds on the level below it. Quality is pursued at the lowest level within the specific fine grained processes. At the next two levels the goal of both SPICE and CMMI is to improve dependability and repeatability, to institute consistent performance across the enterprise. However, neither SPICE nor CMMI have direct goals of software development speed or time to market, nor cost effectiveness through the entire value chain ; they are rather the expected side effects of reducing the sources of error and deviation. Software development is not manufacturing so some caution is needed when drawing lessons from one to apply to the other. However, the general theory of cumulative capabilities does offer some interesting direction to process improvement. Instead of constantly trying to improve software development processes through eliminating software defects, an organization might consider improvements intended to support customer's different needs (flexibility) or improvements that reduce the overall time to develop the software (speed).

Service industries too have offered a theory to explain better performance among service providers. The theory of swift, even flow [25] has been used to explain why some organizations outperform others in their sector. This theory proposes that “productivity of any process rises with the speed by which materials flow through the process and falls with increases in the variability associated either with demand on the process or with the steps in the process itself." [26]. Based on this theory, process improvements should aim to reduce the variability of demand on the process by reducing batch size and increase the speed through the process by reducing delays, work in progress, queues and so on. This supports the general move in software development toward more iterative and agile development methods and away from "big bang" plan-based projects.

Empowering practitioners is a key motivator in process change that reduces resistance towards improvement. At the same time, process change initiated by those who perform the process is not widespread in practice [16]. A way to reduce resistance to change is to demonstrate quickly that the improvement will achieve the desired results. Thus, it can be assumed that the management turns their attention towards areas in which improvements become visible quickly [27]. It remains unknown when and how the management becomes aware of the necessity for process improvement and how they decide to support process improvement while not being sure of its effectiveness [28].

According to these different views of process and process improvements, improvements initiated by developers should favour informal improvements, contained within the process and implemented through the practices of that process. Improvements initiated by project managers or senior managers should favour their management concerns about visible success or ability to meet targets. Improvements that affect the allocation or distribution of resources should be of interest to all of them. Given the different interests of management and process performers there appears to be no reason why informal process improvement practices of the agile community that largely affect the work of developers cannot be used concurrently with the more formal process improvement methods of SPICE or CMMI that largely affect management of work through the processes. Using one process improvement method does not preclude simultaneously using another.

## 3 Research Method

The main purpose of this research was to establish the current state of process improvements in software engineering or service management. Based on the questions to be investigated and a wish to reach as many potential respondents as possible, a survey was considered the most appropriate means to gather data. In an effort to overcome regional biases, participants in this research were sought through professional networks, process improvement user groups and international standards groups. Although this gave a better international distribution, the sampling method used was non-probabilistic which challenges the external validity of the results. In all, 13 researchers from Australia, Canada, Estonia, Finland, France, Germany, Hong Kong, Ireland, Japan, South Africa and South Korea participated. This project used a web-based collaboration tool to distribute draft documents and to discuss topics of interest.

A survey proposal was developed and distributed to the research collaborators for discussion to allow consensus to be reached about the scope of the survey and the research questions to be addressed by the survey. Categories and potential responses to many of the questions were discussed and decided among the participating researchers based on their knowledge and experience of software development, service management and process improvement. The questionnaire was developed to be a survey, rather than an interview which allowed for the widest dissemination and for consistent administration at different locations.

## 4 Research Questions

Process improvement is often considered to be disruptive, expensive and not always successful which would discourage organizations from conducting it. Yet many organizations do attempt to improve their software development or business processes thus there must be some motivation for doing so. To understand the motivation in industry, our first research question was

- **What motivates process improvement?**

There are many ways to identify potential process improvements, from formal assessments and audits, regulatory and business environment changes, and the ever present project failure. Almost any method of identifying improvements will reveal multiple competing potential improvements from which an organization must select a manageable number to implement. While a rational organization might thoroughly review the return on investment, other organizations might simply pick the most obvious or even the most politically sensitive improvement. Process improvement methods generally have something to say about selecting improvements to implement but there is little empirical information about just what organizations do to justify their selection of process improvements, hence our second research question was;

- **What data is sought to justify the improvement?**

Process improvements usually have a goal ranging from the very general goals of improving product quality [29], to meeting the requirements of quality standards [8], or other process models [9, 11], to pursuing an organization strategic objective to very specific problems. To better understand how the motivation, justification and the goals of improvements are aligned, our next research question was;

- **What is the improvement intended to achieve?**

There may be a world of difference between an organization's declared process improvement goals and what changes were actually made to processes, either in pursuit of those goals or independent of those goals or as a side effect of process improvements. A perpetually changing environment is likely to have some effect on processes through possibly small and subtle changes. Certainly regulatory changes can sometime trigger changes to processes but these might not be seen as process improvements. The researchers were aware that not all process changes arose from any form of process improvement initiative and wanted some indication of the types of changes that had been made. Hence our fourth question was;

- **What specific changes are made to the processes or other organizational area?**

Most of the data sought required categorical data so were constructed as a question with the most commonly expected answers listed as possible responses, and the usual "other" response that allowed elaboration by the respondent. There were some questions concerning improvement and the organization that suited Likert scale responses that were grouped into the one section.

The participating researchers were only too aware of the limitations of questionnaires for exploratory research, as opposed to confirmatory research, but elected to use a questionnaire to get some basic data on process improvement in order to guide future, more directed research should this be indicated by the results.

The concept of "process" can mean the entirety of software development activities, from business case development to software delivery, implementation and support or can mean a small unit of collaborative work. For this survey, a process was defined as the latter, a small unit of collaborative work since that is the usage of the term within the ISO standards, SPICE and CMMI communities participating in this survey. However, there were no specific instructions in the survey about the scope or meaning of "process" so it is possible that some respondents assumed a different meaning.

Promotion in the different world regions was left to individual researchers in that region. This allowed researchers to tailor or translate the survey to better suit that region while avoiding duplicated responses. As well, this avoided multiple requests to fill out the same survey that may have resulted if promotion was through an international interest groups or international distribution lists. There was no formal monitoring or directive concerning the survey's distribution or recruitment of respondents but it seems that most of the participating researchers sent the survey to their colleagues or industry contacts. This form of non-probability sampling known as "snowball" technique limits external validity. The survey was open for a period of four months, from September to December 2009, with most responses coming in the first month, very few in the last month.

## 5 Survey Results and Data Analysis

Within the survey very few responses were mandatory, leading to some differences in the number of responses to each question. The exception to this was the mandatory consent to participate in the research was requested of the participants. The data was analyzed with descriptive statistics where the data presented are the actual counts, not percentages. The data analysis results are presented in the sequence of each survey question.

### 5.1 Demographic Information

Information was sought on the type of organization making the improvement, the size of the organization, the respondent's role in the organization and the country or region in which the improvement was implemented (Table 6).

After a period of four months, the total number of completed responses was 80. The responses indicate that most of the organizations were "full service" (58%). Some 20% developed (and transitioned) but did not operate the service once deployed while about 17% of respondent organizations were part of a team or alliance rather than having full control over the development or operation (Table 1).

**Table 1.** Organization type

| | Count | % |
|---|---|---|
| Full service | 46 | 58 |
| Develop and transition | 13 | 16 |
| Develop only | 3 | 4 |
| Alliance partner | 5 | 6 |
| Subcontractor | 9 | 11 |
| Other | 4 | 5 |
| Total | 80 | |

The demographics of organization size (Table 2) indicate that it is the larger organizations that have responded to this survey. It could also indicate that changes to the way work is performed is not considered as process improvement unless there is some formal recognition of "process improvement" such as through a formal process improvement initiative.

**Table 2.** Organization size

| | Count | % |
|---|---|---|
| Micro enterprise (Less than 10 employees) | 8 | 10 |
| Very small enterprise (less than 25 employees) | 2 | 3 |
| Small company (Less than 100 employees) | 11 | 14 |
| Large company (100+ employees) | 32 | 40 |
| Multinational company | 27 | 34 |
| Total | 80 | |

Survey respondents (Table 3) were mostly quality managers or their equivalent or senior managers. From this we expected that a large proportion of implemented improvements would concern process management.

**Table 3.** Respondent's role

| | Count | % |
|---|---|---|
| Senior manager | 20 | 25 |
| Project manager | 5 | 6 |
| Development manager | 5 | 6 |
| Process manager | 7 | 9 |
| Quality manager, SEPG manager etc. | 25 | 31 |
| Technical lead | 13 | 16 |
| Other | 5 | 6 |
| Total | 80 | |

## 5.2 Motivation

Process improvement is usually disruptive and many organizations would prefer to continue working as they have done in the past. To decide to change the way work is done usually requires a strong reason.

The survey sought to answer the question of what motivates process improvement. Formal process improvement is a multi-disciplinary undertaking and progress is possible as long as the correct issues are addressed [30]. We sought to find out these issues that motivate organizations to undertake process improvement.

It is interesting that over a quarter of the respondents (Fig. 1) identify a strategic motivation. In general, motivation can be strategic or tactical. Strategic change stems from a higher level in the organization, tends to be larger in scope, and does not necessarily have a cost/benefit justification. An example of this might be a senior management level decision to change the business model of the organization; the change is seen as necessary for the organizations success or survival. Tactical change stems from a lower level of the organization and tends to be smaller in scope. Tactical changes may be justified as supporting a strategic plan, but usually have a cost/benefit business case to support their proposal.

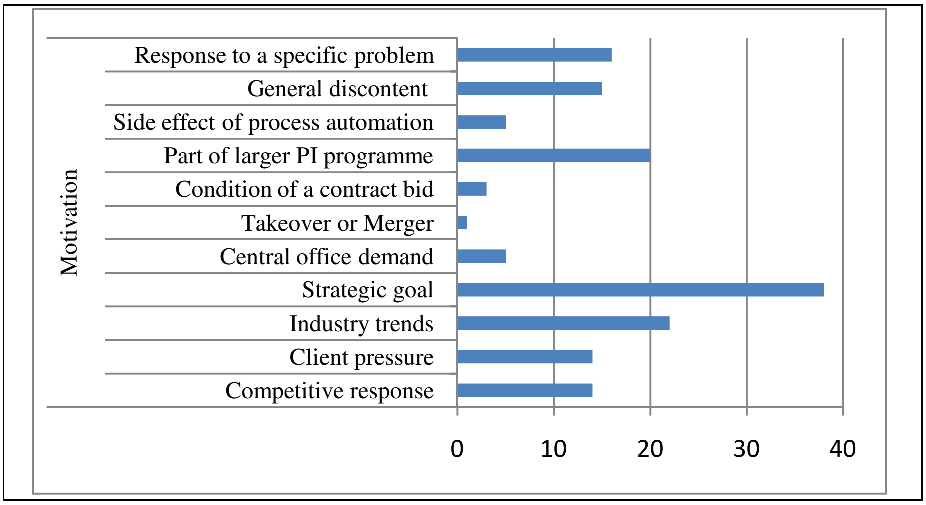

**Fig. 1.** Motivation for initiating process improvements - 153 Responses (Multiple responses allowed)

Since the survey allowed multiple responses it is difficult to tease out the separation of strategic and tactical, but it should be noted that 38 respondents gave "a way of achieving one or more organizational strategic goals" as one of their motivations. While this represents 26% of the motivations cited, it represents nearly 55% of the survey respondents which may indicate that high level management support is either provided or sought. It may also indicate that process improvement has now been accepted, like quality assurance, as an essential activity.

## 5.3 Improvement Initiation

Within a general programme of process improvement, the question of “what initiated a specific improvement” is interesting. The main agent that initiated an improvement appears to have been the process improvement or process management team (Fig. 2). This would indicate that identifying the need for a process improvement is more likely by someone familiar with processes and process improvement.

The large number of responses from people who consider themselves to be part of a process improvement team may reflect the survey distribution, which was through the process improvement professional community, but could also indicate that people with some knowledge of process improvement are more likely to know that the current situation is not inevitable or that organizations with a process improvement team are already convinced of the need for process improvement. Another possible explanation is that extra capacity is needed in order to bring about change [17] with the process improvement team providing that extra capacity.

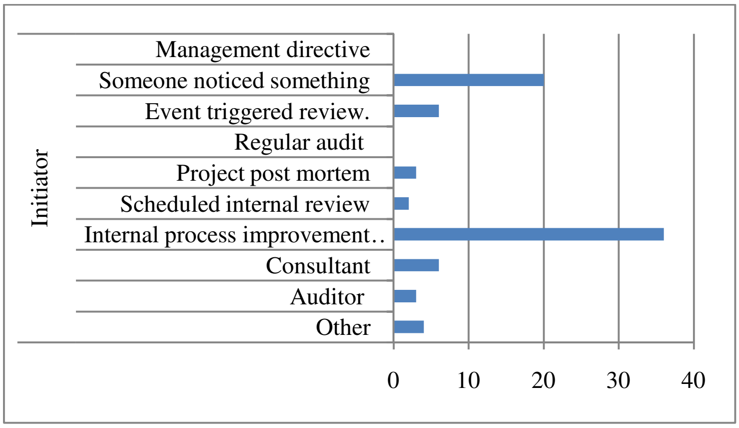

**Fig. 2.** Process improvement initiator -80 Responses (Single response allowed)

## 5.4 Improvement Goals

While there may be many justifications for a process improvement, there is usually one main goal for the actual change. The responses in our study (Fig. 3) indicate that organizations are more concerned about product quality than production costs or competitiveness.

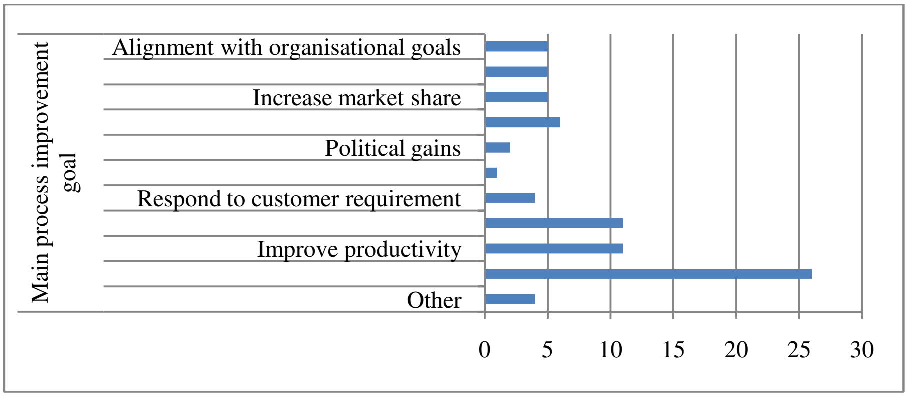

**Fig. 3.** Main process improvement goal - 71 Responses (Single response allowed)

These responses (Fig. 3) also indicate that many organizations have not yet grappled with aligning processes with organizational strategy or that it is not regarded as the most important problem. This is surprising given the general acknowledgement of the importance of aligning processes to organizational goals. Organizations widely regarded as very successful over the long term are usually very good at changing their processes to align with and support their ever changing business processes and strategic goals [30-32]. Again this presents an opportunity for the process improvement community to develop case studies showing how such alignment might be achieved and the gains to be made from doing so. There was no correlation between the role of the respondent and the improvement goal, nor between the improvement initiator and the goal of the improvement. This could indicate that respondents were unaware of the actual goal of the improvement or that there is little understanding of process improvement other than changes to work practices.

## 5.5 Implementing the Improvement

Processes are most often described in terms of work instructions with less attention being paid to other elements of the process. Processes for highly repetitive work may have been such that all the peripheral elements of the process are designed in. However the highly variable processes require those performing the process to make many more decisions about what to do in the specific situation.

Any examination of processes will reveal templates, checklists, the required skills and experience or other social factors that are essential elements of the process. Similarly a process change will seldom be singular, with changes in one place requiring matching changes in another part of the process. The survey sought to establish what specific changes were made to the processes.

**Table 4.** Changes made to improve processes

| **What specific changes were made to the process?** 363 Responses (Multiple responses allowed) | |
|---|---|
| Changed work instructions | 43 |
| Changed presentation of work instructions | 36 |
| Changed order of the work activities | 25 |
| Changed process input work product | 25 |
| Changed work product review criteria | 28 |
| Added templates or changed existing templates | 51 |
| Changed information used by the process | 28 |
| Introduced new or changed techniques | 42 |
| Trained people in work techniques | 9 |
| Introduced new or changed tools | 36 |
| Trained people about the processes | 6 |
| Automated the process | 17 |
| Provided training to improve inter-personal skills | 9 |
| Other | 7 |

Clearly it is pointless to concentrate on changing the work instruction if the most effect can be gained by training people in how to perform those work instructions.

No one change dominates (Table 4. ) but the responses show a tendency to add or change process support elements. An example of a process support element are the templates that are added or changed to help know what is expected or how it is to be presented. Some of these changes will usually be made in combination. The two most popular changes, changed work instructions and changed work techniques, will usually require matching changes in other elements of the process. One of the researchers observed that the process description may be read infrequently but templates are used every time the process is performed, so are more important improvement elements.

## 5.6 Improvement and the Organization

Information was sought on a number of questions about how process improvement is perceived within the organization. Responses (Table 5. ) were on a Likert scale from Strongly agree to Strongly disagree.

Commonly cited beliefs are that process improvement changes are seldom reviewed, are seldom permanent and can make the situation worse. The responses in this survey contradict these assertions, indicating quite emphatically that process improvements are reviewed, do become permanent and seldom make the situation worse.

**Table 5.** Improvement and the organization

| | **% Responses** | | | | | |
|---|---|---|---|---|---|---|
| **Question** | Strongly agree | Agree | Neutral | Disagree | Strongly disagree | N/A |
| This improvement had active senior management support | 41 | 39 | 10 | 0 | 4 | 6 |
| The people who perform the process supported the change | 25 | 51 | 20 | 0 | 3 | 1 |
| The process improvement team supported the change | 50 | 36 | 5 | 3 | 1 | 5 |
| This organisation is concerned about process improvement | 31 | 49 | 10 | 3 | 4 | 4 |
| The improvement was formally reviewed after implementation | 16 | 31 | 23 | 15 | 6 | 9 |
| The improvement achieved all of its intended goals | 18 | 41 | 24 | 9 | 5 | 4 |
| The improvement made the overall situation worse | 5 | 4 | 4 | 28 | 55 | 5 |
| The improvement has become permanent. | 30 | 45 | 11 | 6 | 3 | 5 |

## 5.7 Regional Differences and Process Improvement

Among the researchers there was considerable interest in the possibility of regional differences in approaches to or choices of process improvements. Respondents were asked to nominate the country or region in which the process improvement was implemented rather than their organisation's country (Table 6. ) The intention was to explore whether or not process improvement differed in different parts of the world. The distribution of responses does not match the distribution of software development and service design and delivery worldwide, so the findings of this survey must be treated as indicative only.

Cluster analysis using StatistiXL did not reveal any relationships between any of the variables. Since it was widely expected among the participating researchers that there would be some regional differences, this absence of relationship was unexpected. While it is possible that the survey was insufficiently sensitive to questions of regional differences it is also possible that process improvement, in software development at least, is determined more by the dominant process improvement methods than by regional differences. Process improvement did not develop separately in different regions and then combine globally but rather developed from the same source of quality management [5, 22, 33]. Thus it is not surprising that any regional differences are undetectable.

**Table 6.** Region where the process improvement was implemented

| Country/Region | Count | Country/Region | Count |
|---|---|---|---|
| Asia/Asia Pacific | 2 | International | 2 |
| Australia | 21 | Japan | 7 |
| Canada | 5 | Kingdom of Saudi Arabia | 1 |
| Estonia | 3 | South Africa | 8 |
| Europe | 2 | South Korea | 11 |
| Finland | 7 | Sweden | 1 |
| Germany | 3 | UK | 2 |
| Holland | 1 | USA | 4 |
| Hong Kong | 1 | Vietnam | 1 |
| Indonesia | 1 | | |

## 6 Limitations of the Study

Face and construct validity were addressed through the extensive review of both the survey proposal and the survey questionnaires. Review and comments by non-native English speakers were particularly valued because they exposed linguistic assumptions.

External validity of the data remains weak. Although the survey was conducted internationally, the number of responses from any one country or region is small. Some countries that are reputed to have significant process improvement programs, like India, are not represented at all.

The respondents in this survey represented mostly large or multinational organizations which is not representative of the general population when some 80% of organizations in the world being classified as small or very small enterprises, even among software developers and service providers.

The non-probability sampling method used for data gathering sets yet another limitation as the results of the study are not generalizable to organizations other than the ones that responded to the survey.

Since the time when the survey was conceived the authors have learned more about processes and their improvement, particularly the less formal improvement methods and different models of improved processes or services. The survey has provided data with which more recent concerns can be examined but it is post hoc analysis, limited in its validity. Nevertheless we believe that there is some value in the analysis.

## 7 Discussion

The motivation for improvement came from the strategic level, in most of the cases, while improvements that were implemented targeted specific process support elements on operational level instead. These findings confirm those of Sterman [27] who claimed 13 years ago that the side effects of process improvement are not analyzed before improvements are started. We can see that it is still the case today as organizational motivations for process improvement are different from the effects of those improvements. Although the majority of respondents in the study claimed that they started process improvement as a response to achieve a strategic goal and to increase the quality of the product or service, instead the improvements were made in process support elements on operational level. The aim of an organization improving its processes is to keep commitment and morale high so that the improvements would continue [27], and one of the most powerful motivators for all practitioners is the evidence that process improvements are successful [16]. The improvements in everyday tools like process support elements are visible faster, leading to possible growth of employee motivation to implement further process changes. At the same time it is worrisome that the improvements are not reaching their original goals on the expected strategic level. This might lead to management questioning the justification for future rigorous process improvement programmes that only improve process support elements.

As a result of the study, we can see that the processes with low technical and organizational complexity may improve rapidly as a result of process improvement because they are not strongly coupled with other organizational processes. Sterman [27] argues that the more complex processes of new product development, customer needs assessment and reorientation of strategy involve high technical and organizational complexity and improving these processes affects the whole organizational environment.

Although the survey results indicate that the improvements have made the overall situation of the organization better, the improvements have not resulted in the expected scope and level of change in the organization. It is difficult to justify the continual effort for improvement to the management until the impact of process improvement is felt at the strategic level. The difference between the expected changes and the implemented improvements might also increase resistance to the organizational change from the employees [34].

Process improvement was viewed by the survey respondents as a formal initiative and not as a personal improvement or a team's way of carrying out work. Since there seem to be more changes made to processes than come from formal initiatives just through perpetual changes in the working environment, this indicates that there is a level of change that is so familiar it is ignored. People learn by doing [35] and in that learning they change the process at a level that apparently escapes attention. Yet it is the informal, intrinsically motivated improvements that Deming believed were the more important driver of process and product improvement [36]. While process improvement remains the concern only of a few in the organization, it is only their knowledge, skills and experience that are utilized whereas if process improvement is seen to be everyone's concern and involve everyone, as Deming believed it should be, then significantly more knowledge, skill, and experience are brought to bear on the problem.

## 8 What Have We Learned

The survey that underpins most of this paper was conducted in 2009 and early 2010. Since that time the authors have had many opportunities to learn more about process improvement in general, and software or service management process improvement in particular. So what have we learned?

While process improvement may seem a singular concept with wide consensus about its purpose, there are signs that this is not true, that the debate about agile software development versus plan-based software development has its parallel in process improvement. Software development can be perceived as an act of production or an act of design [37] or at the extreme could be regarded as an act of sensemaking [38]. Each different perception will generate a different goal for process improvement. Improving production processes is different from improving design processes which is different from improving sensemaking processes. The goals of each type of process are different as will be the goals of improving them.

With this recent understanding investigations into process improvement would be careful to distinguish the different types of software development and the different goals of process improvement in order to guide the research and elicit more appropriate and more useful information.

Psychological and organizational studies concerning empowerment of employees at workplace have given some insights about how organizations could build the motivation and commitment in process improvement. We have discovered that employees want to align their processes to organization's strategic goals but they do not know how to do it. Employees' motivation to work towards organization's goals is high after process assessment but falls drastically once the improvements have been implemented [34]. This supports the findings of the study presented in this paper that the original goals for improvements are not always satisfied by the final implemented improvements.

## 9 Summary and Future Works

Process improvement in software development is one form of organizational change in one particular domain that might be compared with general organizational change and change in other domains. However, the concern of this survey and of this paper was to determine what actual changes were made to the processes. Comparing software process improvement and business process improvement could be a fruitful subject for further research. Understanding the complex effects process improvement has in an organizational environment could be useful for the organizations undergoing improvements.

The responses concerning motivations for process improvement were not matched by the responses concerning improvement goals in our study. This survey did not set out to investigate such a possibility and it may have been an effect of the questionnaire construction or of the question wording but could also indicate a very real mismatch. If there is evidence of such a mismatch it would indicate that little is known about how to align processes and process improvement to the organization's strategic goals. This seems to be sufficiently important to warrant further work.

The performance scales of both Schmenner and Ferdows and De Mayer offer interesting alternatives to the frequently assumed improvement goal of productivity and may provide some direction for improved improvement methods. Certainly the theory of swift even flow is a counter argument to the tendency to increasing weight and rigour in processes, since the need for speed will usually require less of both.

**Acknowledgements.** This research project was supported by EDS, an HP company and by the National Research Fund, Luxembourg and cofounded under the Marie Curie Actions of the European Commission (FP7-COFUND).

## References

1. Clark, B.K.: The Effects of Software Process Maturity on Software Development Effort. Faculty of the Graduate School, University of Southern California, San Diego, p. 140 (1997)
2. Goldenson, D.R., Gibson, D.L.: Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results. Software Engineering Institute (2003)
3. Goldenson, D.R., Herbsleb, J.D.: After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success. Software Engineering Institute (1995)
4. Herbsleb, J.D., Goldenson, D.R.: A systematic survey of CMM experience and results. In: Proceedings of the 18th International Conference on Software Engineering 1996, pp. 323–330 (1996)
5. Humphrey, W.S.: A Discipline for Software Engineering. Addison-Wesley (1994)
6. Gilb, T.: Principles of Software Engineering Management. Addison-Wesley (1988)
7. Boehm, B., Clark, B., Horowitz, E., Madachy, R., Selby, R., Westland, C.: An Overview of the Cocomo 2.0 Software Cost Model. In: Software Technology Conference, p. 21 (1995)

8. ISO/IEC 9001: Quality Management Systems - Requirements. 9001 (2008)
9. ISO/IEC 15504-2: Software Engineering - Process Assessment - Part 2: Performing An Assessment. 15504, vol. 2 (2003)
10. Schroeder, R.G., Linderman, K., Liedtke, C., Choo, A.S.: Six Sigma: Definition and underlying theory. Journal of Operations Management 26, 536–554 (2008)
11. SEI: CMMI® for Development, Version 1.3. Software Engineering Institute (2010)
12. Middleton, P.: Lean Software Development: Two Case Studies. Software Quality Journal 9, 241 (2006)
13. Womack, J.P., Jones, D.T., Roos, D.: The Machine that Changed the World. Rawson Associates, New York (1990)
14. Rummler, G.A., Ramias, A.J., Rummler, R.A.: White Space Revisited: Creating Value Through Process. Jossey-Bass, San Francisco (2010)
15. ISO/IEC 12207: Information technology – Software life cycle processes. 12207 (2008)
16. Baddoo, N., Hall, T.: Motivators of Software Process Improvement: an analysis of practitioners' views. The Journal of Systems and Software 62, 85–96 (2002)
17. Hoverstadt, P.: The Fractal Organization: Creating sustainable organizations with the Viable System Model. Wiley (2008)
18. Cockburn, A.: Agile Software Development: The Cooperative Game, 2nd edn. Addison-Wesley Professional (2006)
19. Cohn, M.: Succeeding with Agile: Software Development using Scrum. Addison-Wesley (2010)
20. ISO/IEC 15504-1: Information Technology - Process Assessment - Part 1: Concepts and Vocabulary. 15504, vol. 1 (2004)
21. Salo, O., Abrahamsson, P.: Integrating Agile Software Development and Software Process Improvement: A Longitudinal Study. ISESE, Noosa Heads (2005)
22. Paulk, M.C., Curtis, B., Chrissis, M.B., Webber, C.V.: Capability Maturity Model for Software. Software Engineering Institute (1991)
23. Ferdows, K., De Meyer, A.: Lasting improvements in manufacturing performance: In search of a new theory. Journal of Operations Management 9, 168–184 (1990)
24. Flynn, B.B., Flynn, E.J.: An exploratory study of the nature of cumulative capabilities. Journal of Operations Management 22, 439–457 (2004)
25. Schmenner, R.W.: Service Businesses and Productivity. Decision Sciences 35, 333–347 (2004)
26. Schmenner, R.W.: Looking ahead by looking back: Swift Even Flow in the history of manufacturing. Production and Operations Management 10, 87–96 (2001)
27. Sterman, J., Kofman, F., Repenning, N.: Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement. Management Science 43 (1997)
28. Lepasaar, M., Varkoi, T., Jaakkola, H.: Models and Success Factors of Process Change. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2001. LNCS, vol. 2188, pp. 68–77. Springer, Heidelberg (2001)
29. Deming, W.E.: Out of the Crisis. The MIT Press (1986)
30. Gray, E.M., Smith, W.L.: On the limitations of software process assessment and the recognition of a required re-orientation for global process improvement. Software Quality Journal 7, 21–34 (1998)
31. Davenport, T.H.: Process Innovation: Reengineering Work Through Information Technology. Harvard Business School Press (1993)

32. Brown, S.L., Eisenhardt, K.M.: The Art of Continuous Change: Linking Complexity Theory and Time-paced Evolution in Relentlessly Shifting Organizations. Administration Science Quarterly 42, 1–34 (1997)
33. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Association for Computing Machinery. Communications of the ACM 35, 75–90 (1992)
34. Lepmets, M., Ras, E.: Motivation and Empowerment in Process Improvement. In: O'Connor, R.V., Pries-Heje, J., Messnarz, R. (eds.) EuroSPI 2011. CCIS, vol. 172, pp. 109–120. Springer, Heidelberg (2011)
35. Brown, J.S., Duguid, P.: Creativity Versus Structure: A Useful Tension. MIT Sloan Management Review 42, 93–94 (2001)
36. Hillmer, S., Karney, D.: In support of the assumptions at the foundation of Deming's management theory. Journal of Quality Management 6, 371–400 (2001)
37. McBride, T., Henderson-Sellers, B., Zowghi, D.: Software development as a design or a production project; An empirical study of project monitoring and control. Journal of Enterprise Information Management 20, 70 (2007)
38. Weick, K.: Managing the Unexpected: Complexity as Distributed Sensemaking. In: McDaniel, R., Driebe, D. (eds.) Uncertainty and Surprise in Complex Systems, vol. 4, pp. 51–65. Springer, Heidelberg (2005)

# Towards Efficient Component Performance Analysis in Component Based Architectures

Nabila Salmi[1,2] and Malika Ioualalen[1]

[1] LSI-USTHB - Université des Sciences et Technologie
BP 32, El-Alia, Bab-Ezzouar, 16111 Alger, Algérie
{salmi,ioualalen}@lsi-usthb.dz
[2] LISTIC - Université de Savoie - BP 80449, 74944
Annecy le Vieux Cedex, France

**Abstract.** The desire to bring better quality and higher efficiency in software design has led to the development of *Component Based Systems*. This kind of development has several benefits, however, at the performance level, no guarantees ensure software correctness and good performance properties. To help application designers to meet desired performance of their applications, this paper proposes a modular analysis process that allows to assess independently and efficiently component performances and its impact on a component based architecture. This process is achieved through a modelling phase, based on Stochastic Well-formed Nets (SWN), a high level model of Stochastic Petri nets, and a compositional structured performance evaluation method. It starts from the system definition given in a suitable Architecture Description Language, the targeted component implementation and an "abstract view" of other components, then provides efficiently system performance indexes. The process is illustrated through an application example.

**Keywords:** Performances, Component-Based Systems, SWN, Composition, structured method, modular analysis.

## Introduction

Software products include more and more various components developed by third parties and being assembled together to reach a common goal. Such systems are known as *Component Based Systems (CBS)*. The main objective of this approach is to produce high quality applications with reduced cost, thanks to components reuse, and to make easier maintaining and upgrade. To allow for CBS definition, several *component models* have been introduced, for both academic and industrial purposes. We quote EJB, CCM, COM+/.net, Fractal, JMX, PECOS, Koala,IEC61499,... [26,18,19,20,16,17,2]. For the most of these models, an Architecture Description Language (ADL) [15] allows to describe an assembly of components. From this description, a set of accompanying tools generate the application code and perform some formal verifications such as type compatibilities.

The component based approach is very promising. However, it is difficult to provide software performance guarantees and ensure that the resulting designed system meets the performance expected by users, as the system is built from independently developed components. Usually, application designers rely on careful planning and continuous testing to reach the functional requirements of the end product. Some of the extra functional requirements, such as performance, are often checked during integration or testing at final phase. So, we need methodologies and tools that allow qualitative and quantitative analysis of CBS, to support designers in their activities.

To achieve this goal, several work was proposed mainly for qualitative analysis. [1]uses model checking of Labeled Transition Systems (LTS) to prove temporal logic properties of a Fractal CBS. [9] and [23], are respectively based on hierarchical coloured Petri nets (HCPN) and generalized stochastic nets (GSPN). In contrast, performance analysis is usually carried out with measures on existing systems using performance testing [28], or even by benchmarking during system operation. These technics are very costly, as error correction need redevelopment leading to time consumption and investment loss. We can however note proposals for predictive performance modelling [5,10]. This approach, followed by [29,11], translates architecture designs, mostly given in the UML language into adequate performance models such as Layered Queuing Networks (LQN) [22]. A survey on performance analysis of CBS summarizes most proposals in [14]. Even though, efficient methods are needed for analysis of important architectures, which may lead to state space explosion.

In this context, we proposed in [25,24] an efficient general modelling and "a priori" analysis method of CBS for performance prediction. This method is based on the Stochastic Well-formed Nets (SWN) [4], a special class of Stochastic coloured Petri Nets, widely used for performance analysis of complex symmetrical systems. It builds automatically a global SWN model for the CBS, seen as a composition of SWN models associated to components and their interconnections. Then, it applies a structured analysis method for the computation of performance indexes of the whole system. The main benefits of our previous method is the reduction of analysis complexity in terms of time computation and memory costs.

In this paper, we propose a component performance analysis process, which is a specialization and extension of our previous work defined for performance analysis of a CBS. The added-value of this process is to allow to concentrate on a given component of a component based system, compute its performances and estimate its impact on the whole architecture. As a long-term objective, this will help application designers to make cost-effective decisions in order to meet the desired Quality of Service (QoS) of their applications. The key in this process is to consider the component under study detailed in the scope of a component based architecture, while remaining component interactions are non detailed, then, assess performances of the whole system.

For this purpose, the process starts from the architecture description of the CBS, provided in an ADL, and models the targetted detailed component while adopting abstracted models for other components. Components are modelled with Stochastic Well-formed Net (SWN) [4], a special class of Stochastic coloured Petri Nets, widely used for performance analysis of symmetrical systems. The CBS global model is then derived on the basis of these models. Finally, our structured compositional method is applied to compute performance indices of the whole architecture.

The structure of the paper is as follows. We present in section 1 the main features of a CBS. We also describe a CBS application as an example used along the paper. We follow in section 2 by giving an overview of our previous work. We explain in section 3 our modular analysis process of a CBS, whose application to the example is illustrated in section 3.4. We conclude and give future works in section 4.

## 1 Component Based Systems

### 1.1 Concepts

A software *component* is defined as a unit of composition, provided with contractually specified *interfaces* and explicit context *dependencies* [27]. An interface is an access point to the component, which defines provided or required services. In addition, types, constraints and semantics are defined by the *component model* in order to describe the expected behaviour at runtime.

Interfaces of a component allow to connect it to other components. Consequently, we build a Component-based System by connecting the interfaces of components. These connections are done depending on interactions between components. Generally, two main styles of interactions are defined in component models: synchronous interactions provided by service invocation (such as an RPC or RMI communication), and asynchronous interactions given through notification of events (asynchronous messages). Service invocations take place between a *client* interface requesting a service and a *server* interface providing the service. Besides, event communications are defined between one or more *event source* interfaces generating events and one or several *event sink* interfaces receiving event notifications. The reception of a notification causes the acknowledgment of the reception and execution of a specified reaction called the *handler* of the event. Some event services can use *event channels* for mediating event messages between sources and sinks. An event channel is an entity responsible for registering subscriptions of a specific type of event, receiving events, filtering events according to specific modes, and routing them to the interested sinks.

A component can contain itself a finite number of other interacting components, called *sub-components*, allowing the components to be nested at an arbitrary level. In this case, it is said a *composite* component. At the lowest level, components are said *primitive*. Sometimes, assembling two components may require an adaptation of associated interfaces, whenever these interfaces cannot directly communicate for example. In this case, the adaptation is done with an extra entity, called *connector*, modelling the interaction protocol between the two components.

For each component model, a corresponding *Architecture Description Language (ADL)* allows to describe an assembly of components forming an application. From such a description, a set of tools are used to compile and generate the application code, while checking syntactical and even some semantical properties.

## 1.2 CBS Illustration: Public Internet Access Payment System

The Charles University in Prague, in collaboration with France Telecom, has developed a prototype implementation of a payment system for public Internet access on airports [13]. Clients access the system via a wireless network (e.g WiFi). They are identified by an IP address. Before being able to establish communication with the Internet, they have to authenticate themselves and/or pay for the service. A client can gain access to the Internet using three different ways:

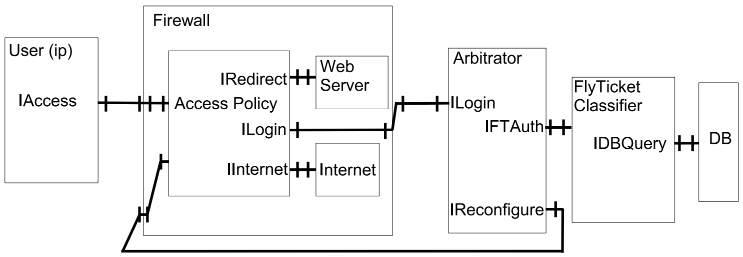

**Fig. 1.** Basic architecture of the payment system

• The client has a valid ticket id for first class or business class and has full internet access during the ticket validity period.
• The client has a valid Frequent Flyer Card and any valid fly ticket. He has also full internet access during the ticket validity period.
• The client can be anyone prepaying Internet access by a credit card.

At the login page, the client is asked a valid ticket id or a frequent flyer id. For authentication process, the system queries the appropriate database for a given ticket id or for a given frequent flyer id. Once authenticated, the user is granted access to any web page until he disconnects or his time-lease expires.

For simplicity, we focus only on a subset of the system. We consider only the first kind of client. Modelling of the behaviour with regard to other clients is similar and can be further studied. So, the architecture of the system of interest (figure 1), described in a Fractal ADL (given by the authors of the paper), consists of :

• A Fractal component user requests.
• A Firewall component which includes an Access Policy, a WebServer and an Internet access sub-components. The Access Policy routes the traffic between the WebServer and the Internet based on the firewall rules.

• An Arbitrator component responsible for the authentication process. It queries a FlyTicket database, and sends back to the firewall the new access policies.
• A FlyTicket Classifier component which queries the airline database appropriate to the requesting ticket id.
• A database component.

## 2 Previous Work

To allow efficient qualitative and performance analysis of CBS, we proposed in [25,24] a general structured compositional method, based on the Stochastic Well-formed Nets (SWN). The main motivations for this formalism is that it is a state based model, making possible the computation of performance indexes, such as the number of requests pending in some part of the system, the mean utilization time of some resource, ... It is also so expressive allowing the modelling of complex systems with concurrency and conflicts. Moreover, interaction between components can be represented with transition/place merging between subnets. Finally, when complex primitive components are involved, high level Petri Nets such as SWNs are almost inevitably required.

### 2.1 Well-Formed Models WN and SWN

The Well-Formed Net (WN) is a high level net. It is a coloured Petri net, where places, transitions and arcs are provided with structured type of tokens and functions.

In this model, elementary entities are modelled with basic classes called *colour classes*. A total order, expressed by a successor function, can be defined on a colour class. A colour class, composed of colours modelling entities of same nature(eg. processes, resources), can be partitioned into static sub-classes, where a sub-class contains colours with identical behaviours, even in terms of performance. Colour classes are brought together to form colour domains (Cartesian product of colour classes) associated to places and transitions. Hence a colour is a tuple of basic colours. The Cartesian product defining a colour domain can be empty (for example, in the case of a place containing neutral tokens). It can also contain repetitions of a class (modelling internal synchronization of this class). Colours of a place label its tokens, whereas colours of a transition define possible firings of the transition. A marking of a place is defined as a multiset (bag) of coloured tokens.

A colour function is attached to each arc and defines for, a given colour of the associated transition, the number of coloured tokens to add or to remove from the attached place. A colour function is built from standard operations (linear combination, composition, ...) of three basic functions. Projection (or identity, denoted by $X$ or $X_i^j$ in figures) selects an element of a tuple; it is represented by a typed variable or by $X$ if no confusion is possible. Synchronization/diffusion (denoted by $S_i$ or $S_{i,k}$) returns the set of all colours of a class ($S_i$) or a subclass ($S_{i,k}$). Successor function is defined for ordered classes only and returns the colour following a given colour.

A transition or an arc function can be guarded by a Boolean expression, a linear combination of atomic predicates. An atomic predicate expresses equality of two variables, or restricts the colour domain of a variable to a static sub-class. A predicate is evaluated on colours instantiated at a transition firing.

The structured definition of a WN allows us to exploit automatically system symmetries, by compacting its reachability graph (RG), leading to a *Symbolic Reachability Graph* (SRG). Nodes of the SRG are *symbolic markings*; each symbolic marking represents a set of ordinary (coloured) markings with equivalent behaviours. Arcs of the SRG stand for sets of equivalent (to within a colour permutation) firings, termed as *symbolic firings*.

Most of the qualitative properties of a WN may be checked directly on its SRG, providing important savings due to the compactness of the SRG with respect to the standard RG when the system exhibits many symmetrical behaviours.

*Stochastic Well-formed Net* (SWN) is the stochastic extension of WN. Each transition is associated with an exponentially distributed delay (depending on the firing colour, and possibly on static sub-classes). The SRG of an SWN, augmented with stochastic firings information, is equivalent to an *aggregated* Markov chain of the chain derived from the coloured net. Thus, main performances indexes of a system can be computed directly from this aggregated chain.

Formal definitions of WN and SWN are given in appendix A and we refer the reader to [4] for more details on SWN.

## 2.2 Overview of Our Method

This method, previously proposed in [25,24] and summarized through figure 2, consists of two main phases :

• A modelling phase : consisting in building automatically a global SWN model for the CBS, seen as a composition of SWN models associated to components and their interconnections, and
• An analysis phase : applying a structured analysis method for the computation of performance indexes.

**Modelling Phase.** This phase starts from the CBS description given with the ADL related to the component model, and translates each component behaviour (source code) and interactions between components into the SWN framework. A component is modelled by an expert by analyzing its source code and modelling activities related to a certain level of details : the expert determine from source code what are main tasks (instructions) to model, depending on the targeted details level, then, given the necessary resources, he models the selected tasks. This is a non trivial and tricky task, that's why an expert knowing the system functioning is required. To avoid to non expert users the modelling task, a repository of SWN models of components can be used. In this case, the user picks, from the repository, models corresponding to the components of his system, provides the cardinalities of modelled entities (colors), such as servers and resources, then, launches the modular analysis.

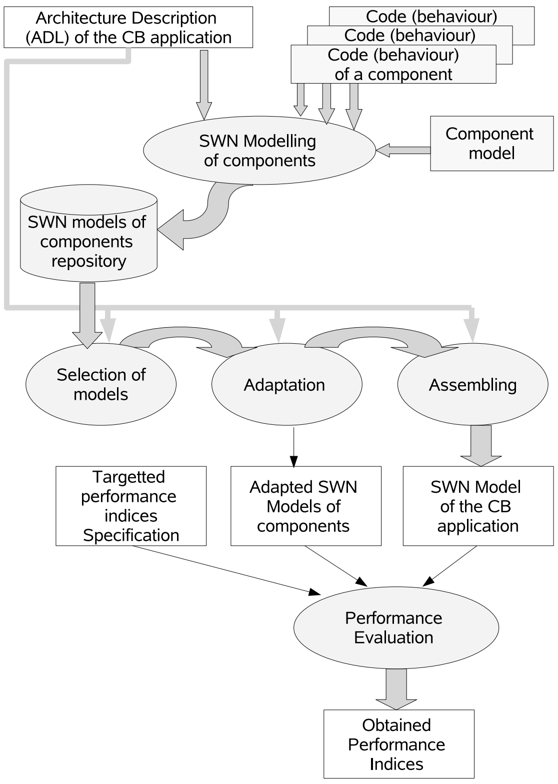
Architecture Description (ADL) of the CB application
Code (behaviour)
Code (behaviour)
Code (behaviour) of a component
SWN Modelling of components
Component model
SWN models of components repository
Selection of models
Adaptation
Assembling
Targetted performance indices Specification
Adapted SWN Models of components
SWN Model of the CB application
Performance Evaluation
Obtained Performance Indices

**Fig. 2.** Principle of compositional analysis approach of CBS

As components communicate through interfaces, a set of rules were proposed for automatic SWN translation. These rules provide generic SWN models for interfaces. In this context, two main components interactions were studied :
(i) A request/response interface, involved in a synchronous method invocation interaction. In this case, the interface is either a *server interface* exposing a service or methods, or a *client interface* invoking a method from a server interface. The *client* is being blocked upon completion of service processing.
(ii) An event based interface, involved in an event based communication. It can be a *source* or *publisher* interface notifying one or more sinks, or a *sink* or *subscriber* interface receiving and processing events. The source resumes its activity while the sinks process the received event. Event messages can also be brokered through an event channel, depending on the specification of the component model.

We illustrate component and interface SWN modelling with figures 3, 4 and 5, showing SWN models built respectively for the Firewall, Arbitror and Classifier components of our CBS example[1]:

[1] Figures obtained using the GreatSPN tool [21].

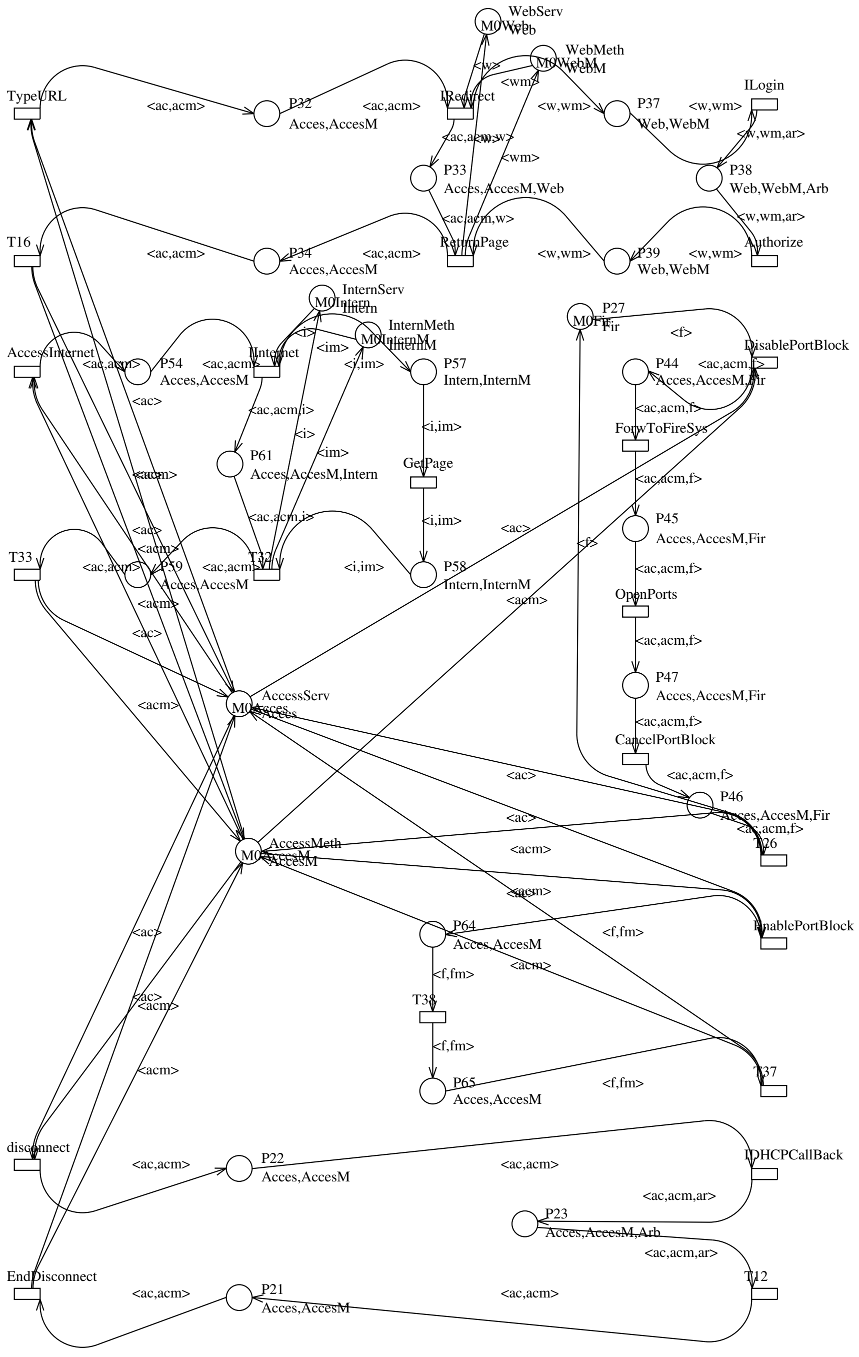

**Fig. 3.** Firewall component SWN model

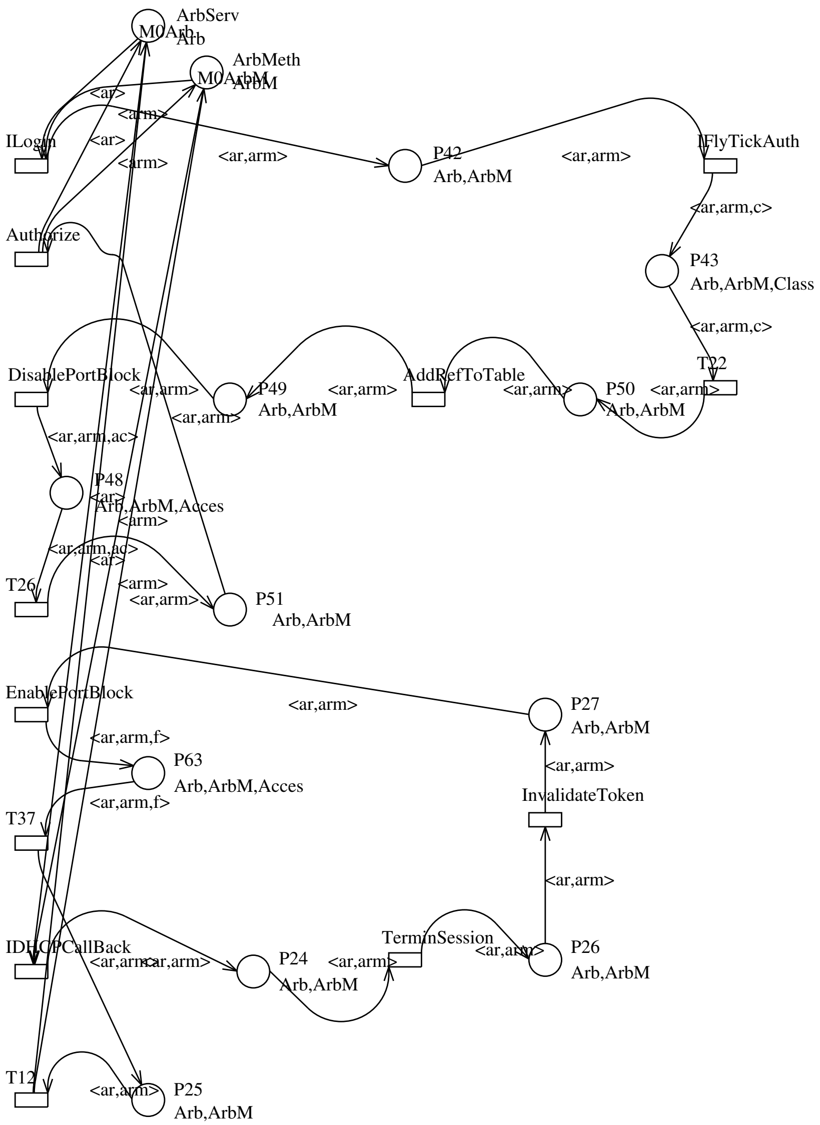

**Fig. 4.** Arbitrator component SWN model

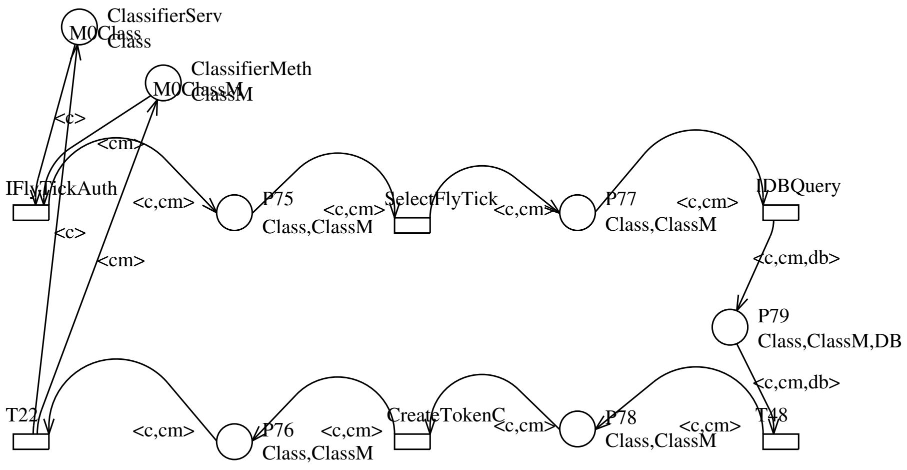

**Fig. 5.** Classifier component SWN model

- The Firewall model exhibits mainly actions triggered when a user types the login url (transition *TypeURL*), accesses Internet (see transition *AccessInternet*), or disconnects (transition *disconnect*) from the system. The Web Server and the Access server sub-components are involved during the first action, whether the Internet access with the Access server sub-components take part in the other actions. In particular, the transition *DisablePortBlock* models the firewall main work which is granting access to the requesting user.
- The Arbitrator model represents the authorization process performed once the login action is called (transition *ILogin*). Other actions are also modelled like enabling port blocking (transition *EnablePortBlack*) or DHCP calling back (transition *IDHCPCallBack*) for terminating a session.
- The Classifier model represents the fly ticket checking validity triggered when asking for authorization (through transition *IFlyTickAuth*).

Once the components and interactions modelled, the obtained models are adapted and modified so that to be composable in the sense of Petri nets composition. The modified models are then composed by merging interfaces elements. We obtain a global model for the CBS, said the *G-SWN*. We keep also the SWN models of components and interactions for the analysis step.

**Analysis Phase.** After generating the G-SWN of a CBS and the SWN models corresponding to its components, we apply the analysis step, allowing to compute performance indexes of the system, such as the response time to a request, the throughput, the mean number of a certain resource... We can also check qualitative properties like the existence of a deadlock between components, the property of reaching a particular state, ...

This is achieved by finding first the set of SWNs representing a possible compatible decomposition of the G-SWN, that fulfill conditions for a structured representation of the SRG and its aggregated generator. Then, we apply a modified version of the structured analysis method presented in [7,8]. The main idea of this method is to study each SWN augmented with "parts" aggregating interactions with other SWN models. These separated studies are then used to derive a tensorial representation [6,3] of the generator of the underlying aggregated Markov chain of the global net (G-SWN), and so to compute performance indexes.

## 3 Modular CBS Analysis Process

We aim at assessing the impact of a given component on the performance of a CBS. This can help to find the most appropriate component parameters that allows to reach a satisfactory level of a targeted CBS quality service.

To reach this objective, our process builds a detailed modelling of the component under study, while keeping an "abstract view" of all other components of the architecture. Then, given the global model of the whole system, it computes the main performance indexes of interest for some given parameters of the component under study. If these indexes don't satisfy the targeted CBS performances, the component parameters are modified and the algorithm is applied once again. This procedure is being repeated as many times as possible until reaching the satisfactory level of the targeted quality of service or performance.

The detailed modelling and the "abstract view" of a component are explained below.

### 3.1 Detailed Component Modelling

The detailed SWN model of a component is built by modelling its internal activities (given a certain details level) and interfaces in the SWN formalism. To achieve that, we proposed in [24,25] systematic rules, mainly devoted to model interfaces involved in communications with other components. Two generic patterns of interaction between components were studied: the synchronous request/response interactions provided with the method or service invocation and the asynchronous interactions consisting of notification of events.

**Definition 1 (Detailed SWN model).** *Let $C$ be a component of a CBS. $N$ is a detailed SWN model for $C$, as described in [25,24] if:*
- *Each service invocation interface (server/client) is modelled according to model of figure 6.*
- *Each event-based interface (publisher/subscriber) is modelled according to model of figure 7.*
- *Internal activities are modelling using SWN concepts.*

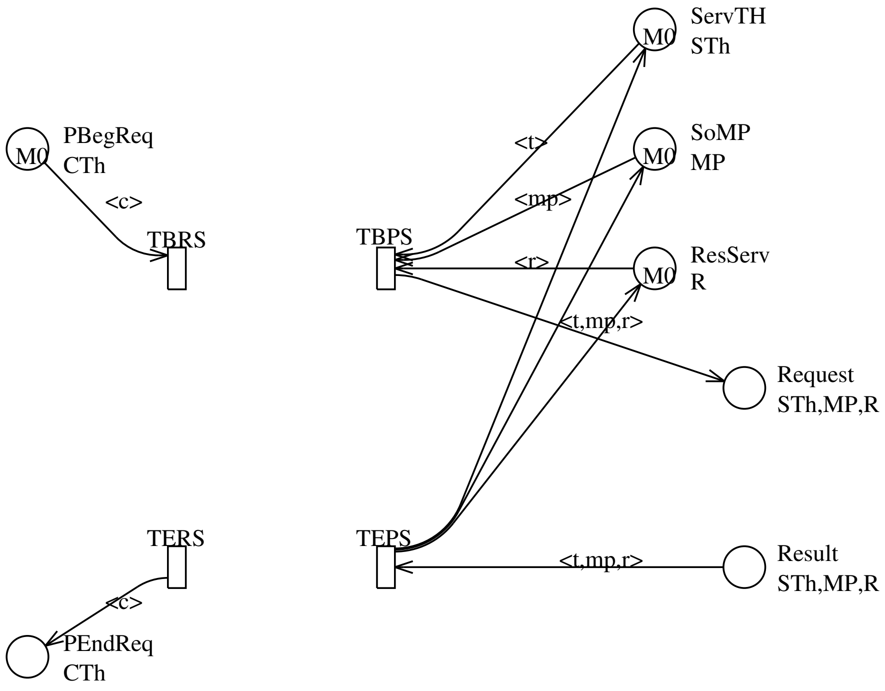

**Fig. 6.** Generic SWNs models of client (left) and server (right) interfaces

## 3.2 Abstract View of a Component

The *abstract view* of a component is a macro representation or *macro-view* that aggregates its behavior while keeping interactions with other components. It is thus a minimal equivalent component modelling with interfaces and interactions modelling.

To build an abstract or macro view, we need to keep a trace of entities being transferred to or from communicating components. These entities correspond, in the case of the SWN model, to colour entities synchronized with other nets. We call them *global colours*. The notion of semiflow is used for computing an abstract view. Let us recall what is a semiflow.

**Definition 2 (Flow and semiflow).** *Let $N$ be a WN with a set $P$ of places, a set $T$ of transitions, and an incidence matrix W=Post-Pre, where Post(p,t), Pre(p,t) for all $p \in P$, $t \in T$ are input and output functions.*

*A flow (resp. semiflow) of $N$ is a vector $f \neq 0$ of $Q^{|P|}$ (resp. $(Q^{+})^{|P|}$) such that $f^T.W = 0$.*

*For all reachable state $M$ : $\sum_{p \in P} f_p.M(p) = \sum_{p \in P} f_p.M_0(p) = \alpha$ (constant), $M_0$ being the initial state of $N$.*

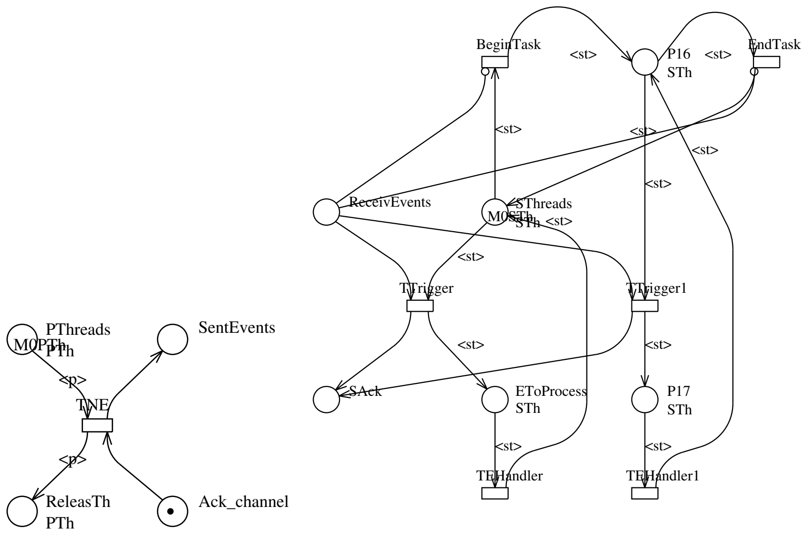

**Fig. 7.** SWN models of event publisher interface (left) and subscriber (right)

The construction of an abstract view follows the guidelines below:

1. Identify the global colours,
2. Find semiflows related to global colours,
3. For each service invocation interface, add the corresponding model.
4. For each event-based interface, add the corresponding model.
5. Connect places of the semiflows to the component interfaces.

The computation of semiflows limited to global colours is done as defined in [12] for an SWN $N$. These semi-flows are said *abstraction semiflows.*

**Definition 3 (Abstraction semiflow).** *$f$ is an abstraction semiflow relative to a colour class $C_i$ of a SWN $\mathcal{N}$ iff $f$ is a semi-flow such that:*
• *The colour domain of $f$ (i.e of places relative to elements of $f$) $\mathcal{D}(f) = C_i$;*
• *$\forall p \in P$, $f$ is 0 or $b$ times a projection ($b$ is a positive constant).*

Given this definition, an abstract view of a component SWN model can be intuitively built by keeping element interfaces required for interconnection with other components models and places containing synchronized entities that are global colours.

Formally, we define an abstract view as follows:

**Definition 4 (Abstract view).**
*Let $\mathcal{N}$ be the SWN model of a component. $\mathcal{N}$ is endowed with:*
• *$k_1$ service invocation interfaces, and*
• *$k_2$ event-based interfaces.*
*$\mathcal{N}$ has a abstract view iff an abstraction semiflow relative to each global class of $\mathcal{N}$ exists.*

*Let* $F = \{f_{C_i} | C_i$ *is a global class of* $N\}$ *the set of abstraction semiflows of* $N$*. The* abstract view *of* $\mathcal{N}$*, denoted* $\mathcal{V}(\mathcal{N})$*, is an SWN with a set* $P_V$ *of places, a set* $T_V$ *of transitions, and input and output functions* $Pre_V$ *and* $Post_V$*, such that :*

- $P_V$ *is the set of places related to semiflows of* $F$ $\{p_f | f \in F\}$*, with :*
  - *The color domain of* $p_f$ $\mathcal{D}(p_f) = \mathcal{D}(f)$*;*
  - *The initial marking of* $p_f$ $M_0(p_f) = \sum_{p \in P} f_p.M_0(p)$*.*

- $T_V$ *is the set of transitions belonging to interfaces.*
- $\forall p \in P_V, \forall t \in T_V$ *:*
  - $Pre_V(p,t) = \sum_{p' \in {}^\bullet t} f_{p'} \circ Pre(p',t)$*, and*
  - $Post_V(p,t) = \sum_{p' \in t^\bullet} f_{p'} \circ Post(p',t)$*.*

Let us illustrate the construction of an abstract view for the Arbitrator component of our example system. This component has two server interfaces connected to the firewall component and three client interfaces, two linked to the firewall and the third to the classifier component. We keep transitions associated to these interfaces. Two global colours are used: *Arb* and *ArbM*. For each global colour, an abstraction semiflow exist, to which we associate an abstraction place (places *Arb_abstract*, *ArbM_abstract*). Linking between these transitions and place according to definition 4 gives the abstract view depicted by figure 8.

## 3.3 Modular Analysis Process Algorithm

Let us consider a CBS $S$ composed of n components $C_1, C_2, \ldots, C_n$, and let us denote $C_x$ the component whose impact on the CBS performance is being investigated. We provide here the general algorithm (Algorithm 3.1), which takes as input components $C_1, C_2, \ldots, C_n$ and different possible parameterizations of $C_x$. Then, it tries to find the most appropriate $C_x$ parameters, which allows to reach a satisfactory level of the targeted performance or quality of service. The algorithm mainly builds SWN models for the different components (detailed model for $C_x$ and abstract views for others), then builds the global SWN (G-SWN) modelling the whole system. It checks then on the G-SWN and separate SWN component models whether conditions for a structured analysis are satisfied on the set of SWN models: if yes, Algorithm 3.2 is applied to compute steady state probabilities and performance indexes; otherwise, the G-SWN is analyzed to get steady state probabilities and performance indexes.

**Deriving the G-SWN.** The global model (G-SWN) of a CBS is derived for two reasons :
(i) for checking satisfaction conditions of our structured method.
(ii) for global system analysis if conditions of the structured method are not satisfied.

The G-SWN is built by interconnecting the detailed SWN model of the component to assess and the abstract views of all others, according to their communication schemas. The communicating interfaces are connected. The obtained

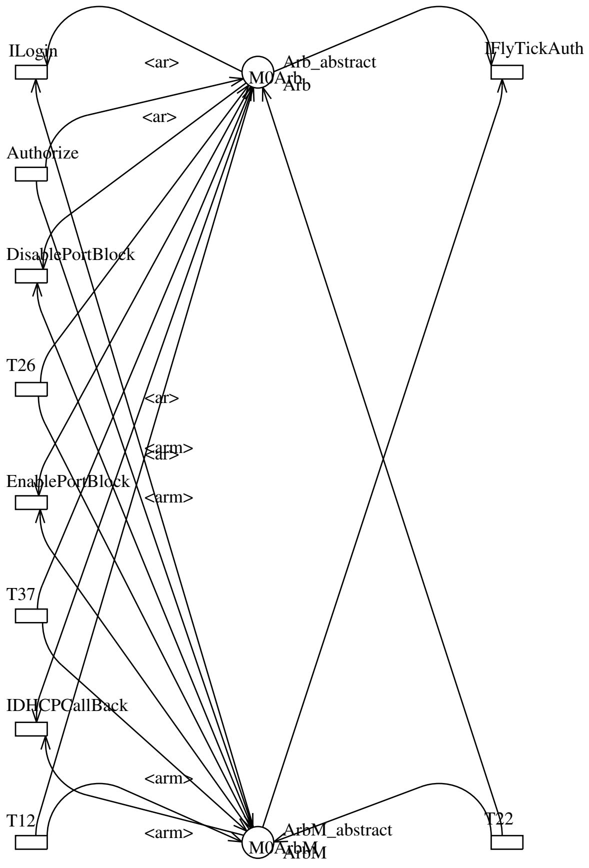

**Fig. 8.** Abstract view of the Arbitrator component

model is then completed by "closing" interfaces of the application with a small Petri net marked adequately, to provide a G-SWN with *finite* state space. This is a classical method, allowing to limit the number of entities in the model and ensure model liveness . In the Petri net context, "closing" means adding a small Petri net to each interface of the application with an adapted initial marking, generally an upper bound of the number of entities.

Notice that only one component modelling is detailed and others are abstracted with abstract views, the associated G-SWN model is composed of several "small nets" with one complete component SWN. However, the efficiency of the structured analysis method depends on the number and the size of SWNs making up the global SWN. So, it is sometimes useful to merge some abstract views into one model for better analysis times.

**Structured Method for Performance Analysis.** After generating the $(SWN_k)_{k \in K}$ corresponding to CBS components, we apply the last step of our

**Algorithm 3.1**

```
$C_1, C_2, \ldots, C_n$ : components of S.
$C_x$ : component to study.
Param(C) : function which parametrizes C with new parameters and returns
           the new parametrized component.
Detail_Modelling(C): function which returns the SWN detailed model of C.
Abstract_Modelling(C): function which returns the SWN abstract model of C.
REPEAT
   $NewC_x$ = Param($C_x$);
   $M_x$ = Detail_Modelling($NewC_x$);
   For $(i = 1; i <= n; i \neq x; i++)$ $M_i$ = Abstract_Modelling($C_i$);
   if (satisfy_conditions(($M_1, M_2, \ldots, M_n$)==True)
      Apply Algorithm 3.2 on the set of SWNs $\{M_1, M_2, \ldots, M_n\}$.
      Get performance indexes (Perf_Indexes) as results of algorithm 3.1.
   else GSWN = Merge($M_1, M_2, \ldots, M_n$);
        Perf_Indexes = Perf_analysis(GSWN);
UNTIL Satisfaction of required criteria on performance indexes Perf_Indexes.
```

analysis approach which computes performance indexes of a system. Examples of performance indexes are the response time of a request, the system throughput, the mean number of a certain resource...

For this purpose, we apply algorithm 3.1, defining a structured analysis method, extended for analysis of CBS in [25,24]. The main idea of the structured method is to study each obtained SWN model augmented with "parts" abstracting interactions with other SWNs. These separated studies are used to derive a tensorial representation of the generator of the underlying aggregated Markov chain of the global net. The tensorial representation is then used to compute steady state probabilities providing important memory usage and computation time savings. These probabilities are used to compute performance indexes.

**Algorithm 3.2**

```
Let $(SWN_k)_{k \in K}$ be SWN models corresponding to the detailed and abstracted
component models of the CBS under study.
1. Extension of the SWNs $\mathcal{N}_k$ to autonomous SWNs $\bar{\mathcal{N}}_k$, said extended nets.
2. Generation of the SRGs of these extended SWNs.
3. Computation of the synchronized product of these SRGs and of the tensorial
   representation of the generator of the underlying aggregated Markov chain.
4. Computation of the steady state distribution of the aggregated model and
   computation of the required performance indexes.
5. Expression of the results in the initial context of the components.
```

Algorithm 3.2 allows important savings of memory usage and computation time, using the structured analysis implemented in our tool *compSWN* [7,8], applied on separate SWN models.

**Table 1.** Transition rates of the studied configuration

| Transition | Rate value |
|---|---:|
| TypeURL | 0.8 |
| IRedirect | 0.9 |
| AccessInernet | 0.9 |
| GetPage | 0.6 |
| IInternet | 0.85 |
| OpenPorts | 0.75 |
| EnablePortBlock | 0.75 |

However, if conditions for a structured analysis are not satisfied, the analysis is performed on the G-SWN modelling the whole CBS, instead of working on the SWN component models, leading to a longer computation time and great memory space, and sometimes, for an important model, the analysis is impossible. The GreatSPN tool [21] is used for the analysis purpose operating on the G-SWN.

## 3.4 Illustration

Let us illustrate our modular process by studying the impact of the firewall component on the Public Internet access payment system presented in section 1.2. The interest is to check whether the number of servers providing the firewall functionalities are sufficient for an acceptable user request response time. To do so, we study the variation of the response time for a client requesting access to Internet, with respect to the firewall processing rate, allowing to grant access to the requesting user.

Thus, the main component which we aim to study is the firewall component.

**Parameters of the System.** According to our modular process (Algorithm 3.1), we take a detailed view of the firewall component (figure 3) and build abstract views for the other components. Before analyzing targetted performances with our method, we need to check conditions for a structured analysis on the global model (G-SWN) modelling the whole system. So, we compose the detailed view of the firewall component and abstract views of other components to get the G-SWN. As, the obtained G-SWN satisfies these conditions, we can apply our structured analysis method following algorithm 3.2. This is done using our tool *compSWN* on subnets which computes steady-state probabilities.

So, we fixed cardinalities of basic colours to 100 clients with 2 firewall servers (modelled by the $Fir$ colour class), 2 arbitrators ($Arb$ colour class), 2 Fly ticket classifiers ($Class$ colour class) and 2 databases ($DB$ colour class). We also take firing rate values of a critical set of transitions (see table 1). Transitions not appearing in this table have rate equal to 1, i.e. faster than all other transitions, rates being given in the same unit. Then, we vary the *DisablePortBlock* transition rate, and study the evolution of response time from obtained steady-state probabilities. We quote that a colour may model a group of elementary entities, for instance a client colour can stand for 100 or 1000 clients. In this case, firing rates of transitions involving this colour should be adapted to the semantics of a colour (100 clients provide a 100 times slower method request rate for instance).

To evaluate the analysis process duration, we mainly focus here on the step computing the steady-state probabilities, as it is the most important step. We evaluated the duration of the modular approach (structured analysis) used for this step and compare it with the usual approach operating directly on the G-SWN for several configurations. The structured analysis proved to be more efficient and provides important savings both in memory usage and computation times, compared to the usual approach applied using the GreatSPN environment. Some configurations cannot also be analyzed with the usual approach, while the analysis was possible with the structured approach.

**Table 2.** State space sizes and computation times for steady-state analysis of the example for various configurations

| Config | —Adr— | —Fir— | —TId— | —Arb— | —Fly— | NbS | NbO |
|---:|---:|---:|---:|---:|---:|---:|---:|
| 1 | 2 | 2 | 2 | 2 | 2 | 38508 | 617952 |
| 2 | 2 | 2 | 2 | 3 | 4 | 90520 | 7886592 |
| 3 | 3 | 2 | 2 | 2 | 2 | 176964 | 8353632 |
| 4 | 3 | 3 | 2 | 2 | 2 | 468432 | 59261888 |

To give an idea of the analyzed state space size, we report in table 2 state space size for some configurations of the whole system. Notations are the following: —Color— is the cardinality of the static subclass, NbS is the number of symbolic markings and NbO is the number of ordinary markings.

Note that for huge models, the two approaches can fail to compute the steady-state probabilities.

**Performance Results.** To reach our goal of studying the impact of the firewall component, we compute several request response times (thanks to the obtained steady-state probabilities), and then compute the variation of the response time with respect to the firewall processing rate, using the rate of the $DisablePortBlock$ transition which models the access granting action.

Figure 9 shows corresponding response time variation results computed first for the configuration of two firewall servers in the firewall component, then for six firewall servers. As the figure shows, the obtained response time is being improved as the firewall processing rate gets bigger (blue curve). However, it is greater than 287s (4,78mn) which is considered as a bad response time for a client accessing to Internet. It is so still poor with respect to the client expectation. Thus, we changed the configuration of the firewall component to six servers and repeat our modular process. We obtain a much better and acceptable response times (red curve), reaching a satisfactory level of the whole system quality of service.

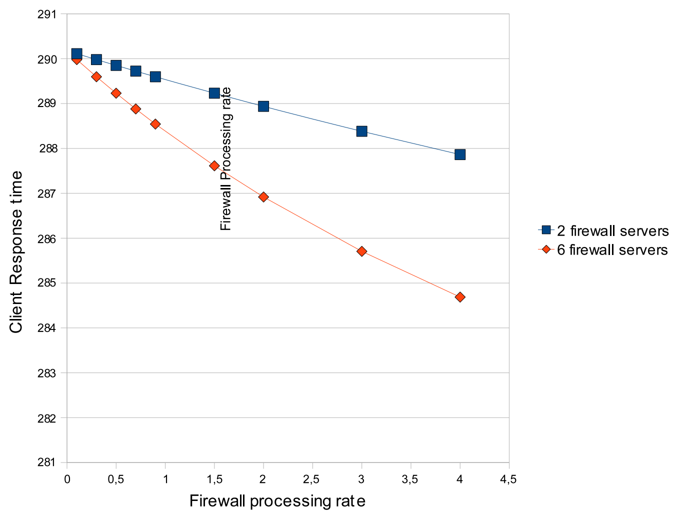

**Fig. 9.** Client response time versus firewall processing rate

## 4 Conclusion

This paper addresses performance modelling and analysis of systems composed of several components, while focusing on a given component. Our goal is to be able to predict the performance of the component on which the study is focused, and estimate its impact on the CBS architecture. For this purpose, we have proposed an efficient component analysis process, allowing to assess "a priori" independently performances of a critical component and predict its behavior. Prediction is based on modelling techniques, allowing to avoid benchmarking and performance testing which is costly and time consuming. The efficiency of the proposed process is derived from that's of the structured analysis method, and consists in important time and memory savings especially when important CBS architectures are addressed.

The process starts from the description of a CB designed application, given in an Architecture Description Language. It models the critical component under study with a detailed SWN model and aggregates the behavior of all other components by building an abstract view for each one. The obtained SWNs are then connected by fusing their interface elements. The resulted G-SWN model is analyzed using our previous structured method.

We tested our proposed process on an example of public internet access payment system. The first results are very promising. But, they still require more research work in several directions, such as automating the construction of a detailed SWN model (which should be proper to each component model) and the derivation of the abstract view of a component.

## References

1. Barros, T., Cansado, A., Madelaine, E., Rivera, M.: Model checking distributed components: The Vercors platform. In: 3rd Workshop on FACS. ENTCS (September 2006)
2. Bruneton, E., Coupaye, T., Stefani, J.B.: The fractal component model, version 2.0-3. Technical report, Fractal team (October 2006) (February 2004), http://fractal.objectweb.org/specification/
3. Buchholz, P., Ciardo, G., Donatelli, S., Kemper, P.: Complexity of Kronecker operations on sparse matrices with applications to solution of Markov models. Technical report 97-66, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA (1997)
4. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans. on Comp. 42(11), 1343–1360 (1993)
5. Smith, C.U., Williams, L.G.: Performance Solutions. Addison-Wesley (2002)
6. Davio, M.: Kronecker products and shuffle algebra. IEEE Transactions on Computers 30(2), 116–125 (1981)
7. Delamare, C., Gardan, Y., Moreaux, P.: Efficient implementation for performance evaluation of synchronous decomposition of high level stochastic Petri nets. In: Proc. of the ICALP 2003, Eindhoven, Holland, June 21-22, pp. 164–183. University of Dortmund, Germany (2003)
8. Delamare, C., Gardan, Y., Moreaux, P.: Performance evaluation with asynchronously decomposable SWN: implementation and case study. In: Proc. of the 10th Int. Workshop on PNPM 2003, Urbana-Champaign, IL, USA, September 2–5, pp. 20–29. IEEE Comp. Soc. Press (2003)
9. Dias da Silva, L., Perkusich, A.: Composition of software artifacts modelled using colored Petri nets. Science of Computer Programming 56(1-2), 171–189 (2005)
10. Firus, V., Becker, S.: Towards performance evaluation of component based software architectures. In: Proc. of FESCA 2004 (2004)
11. Grassi, V., Mirandola, R., Sabetta, A.: Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. J. Syst. Softw. 80(4), 528–558 (2007)
12. Haddad, S., Moreaux, P.: Asynchronous composition of high level Petri nets: a quantitative approach. In: Billington, J., Reisig, W. (eds.) ICATPN 1996. LNCS, vol. 1091, pp. 193–211. Springer, Heidelberg (1996)
13. Jezek, P., Kofron, J., Plasil, F.: Model checking of component behavior specification: A real life experience. Electronic Notes in Theoretical Computer Science 160, 197–210 (2006)
14. Koziolek, H.: Performance evaluation of component-based software systems: A survey. Performance Evaluation, Special Issue on Software and Performance (August 2010)
15. Medvidović, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Trans. on Soft. Eng. 26, 70–93 (2000)
16. Microsoft. Net 3.0 framework (July 2007), http://msdn.microsoft.com/netframework
17. Müller, P., Stich, C., Zeidler, C.: Components@work: Component technology for embedded systems. In: Proc. of the Component-Based Software Engineering Track at the 27th IEEE Euromicro Conference (Euromicro CBSE 2001) (September 2001)

18. Object Management Group. Common object request broker architecture (CORBA) - specification, version 3.1, part 1: CORBA interoperability (July 2007) (2004), http://www.omg.org/cgi-bin/doc?pas/04-08-01.pdf
19. Object Management Group. Common object request broker architecture (CORBA) - specification, version 3.1, part 2: CORBA interfaces (July 2007) (2004), http://www.omg.org/cgi-bin/doc?pas/04-08-02.pdf
20. Object Management Group. CORBA component model specification. version 4.0 (April 2007) (April 2006), http://www.omg.org/cgi-bin/apps/doc?formal/06-04-01.pdf
21. Perf. Eval. Group. GreatSPN home page (2002), http://www.di.unito.it/~greatspn
22. Petriu, D., Shousha, C., Jalnapurkar, A.: Architecture-based performance analysis applied to a telecommunication system. IEEE Transactions on Software Engineering 26(11), 1049–1065 (2000)
23. Rugina, A.E., Kanoun, K., Kaaniche, M.: A system dependability modeling framework using AADL and GSPNs. Technical Report 05666, LAAS (November 2006)
24. Salmi, N., Moreaux, P., Ioualalen, M.: From architectural design to swn models for compositional performance analysis of component based systems: application to ccm based systems. In: Proc. of the 24th UKPEW 2008 Performance Engineering Workshop, pp. 123–136. Imperial College, London, UK (2008)
25. Salmi, N., Moreaux, P., Ioualalen, M.: Performance evaluation of fractal component based systems. Annals of Telecommunications. Special issue: Software component: The Fractal Initiative 64(1), 81–101 (2009)
26. Sun Microsystems. EJB 3.0 specification (July 2007), http://java.sun.com/products/ejb/docs.html
27. Szyperski, C.: Component technology - what, where, and how? In: Proc. 25th Int. Conf. on Software Engineering, May 3-10, pp. 684–693. IEEE (2003)
28. Weyuker, E., Vokolos, F.: Experience with performance testing of software systems: issues, an approach and case study. IEEE Transactions on Software Engineering 26(12), 1147–1156 (2000)
29. Wu, X., Woodside, M.: Performance modeling from software components. SIGSOFT Softw. Eng. Notes 29(1), 290–301 (2004)

## A WN and SWN Formal Definitions

We remind the reader with the definitions of WN and SWN. A detailed presentation of these models can be found in [**?**].

**Definition 5 (Well-formed Petri Net (WN)).** *A well-formed Petri Net S is a tuple* $(P, T, C, cd, Pre, Post, Inh, Guard, Pri, M_0)$ *with:*

- $P, T$*: the finite sets of places and transitions,*
- $C = \{C_i / i \in I = \{1, \cdots, n\}\}$*: the set of basic colour classes;* $C_i$ *is possibly partitioned into into* $n_i$ *static sub-classes:* $C_i = \bigcup_{j=1}^{n_i} C_{i,j}$*,*
- *cd:* $P \bigcup T \rightarrow Bag(I)$*.* $cd(r) = C_1^{e_1} \times C_2^{e_2} \times \ldots \times C_n^{e_n}$ *is the colour domain of a node* $r$*;* $e_i \in \mathbb{N}$ *is the number of occurrences of* $C_i$ *in the colour domain of* $r$*, where Bag(I) is the set of multisets (bags) on I.*
- $Pre, Post, Inh$*: the input, output and inhibition standard colour functions from* $C(t)$ *to* $Bag(C(p))$*.*

- *$Guard(t)$ : $C(t)$ $\rightarrow\{true, false\}$ is a standard predicate associated with the transition $t$. By default, $Guard(t)$ is the constant function of value True.*
- *$Pri : T \rightarrow \mathbb{N}$ the priority function. By default, we assume $\forall t \in T, Pri(t) = 0$;*
- *$M_0 : M_0(p) \in Bag(C(p))$ is the initial marking of $p$.*

**Definition 6 (Stochastic Well-formed Net (SWN)).** *A Stochastic Well-formed Net is a pair $(S, \theta)$ such that:*

- *$S$ is a Well-Formed Net.*
- *$\theta$ a function defined on $T$ such that: $\theta(t) : \tilde{cd}(t) \times \prod_{p\in P} Bag(\tilde{C}(p)) \longrightarrow R^+$.*

*$\theta(t)(\tilde{c}, \tilde{M})$ represents:*

- *The weight of $t$ for the colour $c$ in the marking $M$, if $\pi(t) > 0$ (t is immediate). the firing probability of $t(c)$ in $M$ is then: $\frac{\theta(t)(\tilde{c},\tilde{M})}{\sum_{(t',c'),M[t'(c')>} \theta(t')(\tilde{c'},\tilde{M})}$.*
- *The firing rate of $t$ for the colour $c$ in $M$, if $\pi(t) = 0$ (t is timed): the enabling duration before the firing of $t(c, M)$ follows an exponential probability distribution with mean $\theta(t)(\tilde{c}, \tilde{M})$.*

In this definition, $\tilde{c}$ is the representation of the colour $c$ in terms of static sub-classes, and $\tilde{M}(p)$ is the representation of the symbolic marking of $p$ in terms of tuples of static sub-classes. $\theta(t)$ depends only on static sub-classes of concerned colours.

# A Case Study on Software Risk Analysis in Medical Device Development

Christin Lindholm, Jesper Pedersen Notander, and Martin Höst

Software Engineering Research Group, Department of Computer Science,
Lund University, Faculty of Engineering,
Box 118, 221 00 Lund, Sweden
{christin.lindholm,jesper.notander,martin.host}@cs.lth.se

**Abstract.** Software failures in medical devices can lead to catastrophic situations. Therefore is it crucial to handle software related risks when developing medical devices. This paper presents the experiences gained from an ongoing case study with a medical device development organisation. This part of the study focuses on the two first steps of the risk management process, i.e. risk identification and risk analysis. The research is conducted as action research, with the aim of analysing and giving input to the organisation's introduction of a software risk management process. The risk identification activities focus on user risks based on scenarios describing the expected use of the medical device in its target environment. Challenging problems have been found in the risk management process with respect to definition of the system boundary and system context, the use of scenarios as input to the risk identification and estimation of detectability used during risk assessment.

**Keywords:** risk management, risk analysis, software development, medical device development.

## 1 Introduction

Software has for many years been an important part of larger systems, such as automotive, telecommunication and finance. The amount of software in health care applications is also increasing. Both IT systems used in the medical domain, like administrative systems, and medical devices like measurement systems, are becoming increasingly implemented in software. Medical devices, which this paper focuses on, often include a large amount of software that is crucial for the functionality of the devices. Examples of devices are monitoring devices for blood pressure, or other monitoring devices, and also actuating devices affecting the human body.

Both monitoring devices and actuating devices can be safety critical, and thus a structured development process is required in order to be able to demonstrate safety and quality of the devices. Standards for this include for example IEC 61508 for development of new hardware and software, and IEC 61511 for integration of components developed according to IEC 61508 [1]. Even if standards are available there is still a need to investigate in more detail how development of software can be carried out with this type of requirements.

The focus of this paper is on risk management, which is an important part of a development process for safety critical systems. Since this is an important area, which is affected by the fact that much of the functionality is implemented in software, there is a need to conduct research on how the steps of this process should be conducted.

Risk management includes identification of risks, analysis and prioritization of risks, and handling and monitoring of risks. In these phases there is not only a need to understand a complex product, but also to understand the complex usage of it. This means that it is necessary to involve several different roles in the work, such as domain experts, technical experts and process experts. In this study medical physicians with competence on the monitored medical processes are involved, together with engineers with competence on the software and hardware and personnel with competence on the required procedures in the organization. The objective of the presented research is to summarize experiences from conducting risk identification and risk analysis in the development of a medical device. This is achieved by participating in these steps in a case study in an ongoing development project.

An earlier version of this paper was presented at the EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) 2011 [2].

In Section 2, background and related work is presented. In Section 3 the case study research method is presented, while the results are presented in Section 4. These results are discussed in Section 5, and the main conclusions are summarized in Section 6.

## 2 Background and Related Work

The medical device domain is a complex domain for many reasons. The working environment is one reason, personnel in healthcare are to a great extent mobile when performing patient work, they are often interrupted and unexpected situations occur during work with patients and medical equipment. Garde and Knaup [3] have identified several domain characteristics that contributes to the complexity for example the product of healthcare, meaning the treated patient has for all practical purpose an unlimited set of characteristics which constantly change and interact. This makes it impossible to categorise patients as products can be categorized. Another characteristic mentioned by them is that the majority of stakeholders are non-technical professionals, for example physicians, nurses and administrators. The multitude of medical standards and medical terminology is another complicating factor that contributes to complexity and have to be considered when working in the domain.

A risk is "the probability of incurring a loss or enduring a negative impact" [4] and in the medical device area it is crucial that the medical device, not in any way constitutes any risks. The medical device development organisations have to address different risks regarding patients, users, environment and third parties (for example service technicians) [5]. A fault or mistake in the medical device domain can mean the difference between life and death. The use of medical software is an inherent risk to patients, medical personnel and surroundings. A goal for an organisation that develops medical software is to be able to identify a set of risks that is as complete as possible. A so complete set of risks as possible gives the organisation the opportunity to take action against the risks in order to avoid them or to reduce the consequences of them. Another challenge for an organisation is that the software in a medical device needs to comply with the same laws and regulations as the medical device itself. How strict and detailed it has to be depends on the product, and there are different laws and regulations to follow depending in what part of the world the product is manufactured.

Risk management must be included in the development process for a medical device according to European and American law [6, 7], and there are also standards that the organisation needs to follow. Concerning risk management for medical devices ISO 14971 (www.iso.org) needs to be considered. This standard defines the majority of the risk management terms and gives a framework for a risk management process without specifying details about how things should be done.

Risk management is an organised process for identifying and managing risks [8]. The risk management process is often divided into four steps displayed in Figure 1.

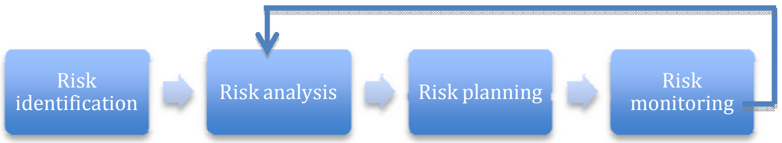

**Fig. 1.** Risk management process

The risk management process for a medical device development organisation must cover all four steps. The research presented in this paper focuses on the two first steps in the process, i.e. risk identification and risk analysis. The reason for this is that these steps are important in the first part developing a complete risk management process with focus on detailed description of each step.

There is a shortage of documented research on software risk management processes in the medical device area and a need for research on several steps in the process at a more detailed level has been identified. The published research covers often the whole risk management process on a high level, not specifically described step by step. McCaffery et al. [9, 10] have developed and tested a software process improvement risk management model (Risk Management Capability Model) that integrates regulatory medical device risk management requirements with the goals and practices of the Capability Maturity Model Integration (CMMI). Schmuland [11] also investigates the whole risk management process, although he focuses on residual risks, i.e. the remaining risks after the risks have been handled, and how to assess the overall residual risk of a product. It is based on the identification of all the important scenarios. There are some researchers that focus on one of the steps in the risk management process for example Sayre et al. [12] who look especially on the risk analysis step. They describe an analytical tool for risk analysis of medical device systems, a Markov chain based safety model and they argue that this safety model presents significant opportunities for quantitative analysis of aspects of system safety.

## 3 Case Study Methodology

The research in this paper is based on a study of a single case. According to Yin "a case study is an empirical inquire that investigates a contemporary phenomenon within its real-life context, specially when the boundaries between the phenomenon and context are not clearly evident" [13]. In software engineering, process improvement activities are often of a complex nature and cannot be studied in isolation, which means that there is a need for empirical studies in real world settings like in this study. The research design of a case study is flexible where the research strategy develops during the data collection and analysis [14].

In action research there is collaboration between researchers and those who are the focus of the research [14]. The observations in this study have been active observations meaning that the researchers have been allowed to influence the outcome of the observed activity. The aim has been to observe how the activities are performed in their context, not to actually perform the activities but to give input and support. The purpose of the active observations probably could be to get interesting aspects of the activities by asking questions and giving advice on relevant topics.

### 3.1 Objectives

The objective of the research in this paper is to give input to the development of a software risk management process in an organisation that develops medical devices. The organisation has a defined risk management process for development of hardware but a need for a risk management process adapted to software development. More specific research questions are:

- RQ1: What are the experiences from using the chosen risk identification method?
- RQ2: What are the experiences from using the chosen risk analysis method?
- RQ3: What are the experiences from focusing on a sub system as a part of a larger system?

That is, RQ1 and RQ2 are general questions about the defined risk management process with focus on pros and cons, while RQ3 was defined based on the architecture of the product that was analysed. The software risk management process focuses only on the development of software for a new medical device (bedside monitor). In this case the new device can be regarded as a sub system since it is a part of a lager system for example the new device import blood pressure values from a patient monitor. The defined risk management risk identification method and risk analysis method is presented in Chapter 4.

## 3.2 Case Study Process

The research process is based on the case study process described by Runeson and Höst [15]. The process in Figure 2 was followed.

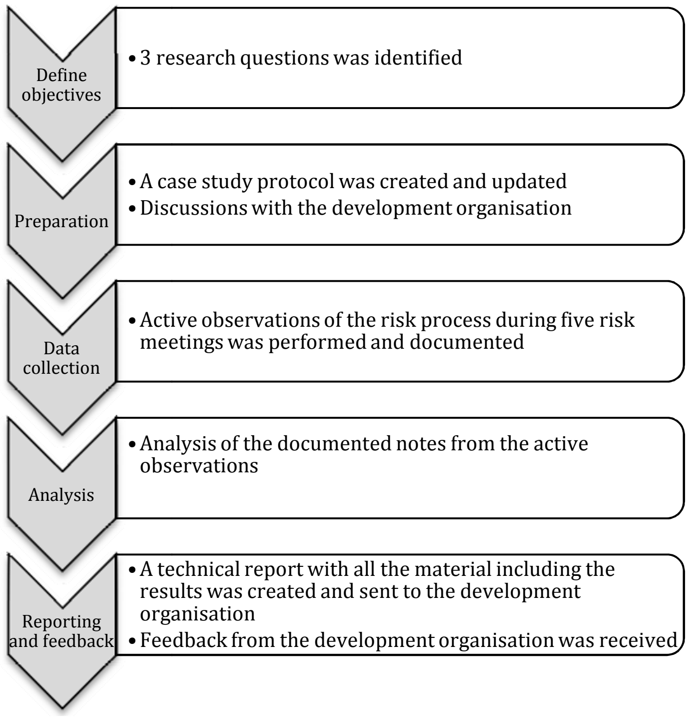
Define objectives
• 3 research questions was identified
Preparation
• A case study protocol was created and updated
• Discussions with the development organisation
Data collection
• Active observations of the risk process during five risk meetings was performed and documented
Analysis
• Analysis of the documented notes from the active observations
Reporting and feedback
• A technical report with all the material including the results was created and sent to the development organisation
• Feedback from the development organisation was received

**Fig. 2.** Case study process

Foremost in the case study the objectives were defined and the preparations were made. As part of the preparations, discussion started with the organisation regarding different risk identification techniques, risk analysis methods and scales for risk classification. The development organisation decided then on the design of the software risk process for the first two steps, i.e. risk identification and risk analysis. The software risk process was then used during five risk meetings with the researchers as active observers of the process. The data collection was made through active observations and the observations were documented, analysed and reported back to the organisation in form of a technical report.

## 3.3 Case Study Context and Subjects

The case study was conducted at a department at a large hospital developing and maintaining medical devices and was performed in different steps during the summer and autumn in 2010. The risk analysis was carried out on a patient monitor system for monitoring intracranial pressure and calculating the cerebral blood flow, including both software and hardware, however the risk analysis was focused on the software. The software in this case is the software developed for the new device (bedside monitor) and the risk identification activities focus on user risks. The purpose of the patient monitor system is to monitor the patients' intracranial pressure, calculate the cerebral blood flow and present it to the medical personal. The patient monitor system consists of mainly three parts:

1. Pressure sensor placed in the patients skull
2. Patient monitor connected to the sensor. The monitor presents and exports blood pressure values
3. Bedside monitor, the new device which import the blood pressure values from the monitor and calculate the cerebral blood flow

Part 1 and 2 of the system have been used before while part 3 is new.

The bedside monitor contains a PC with a Windows operating system and an application based on Palcom and written in Java. The graphical interface presents the calculated cerebral blood flow and the measured intracranial blood pressure.

Participants in the study represent three different groups:

- the intended users with special domain knowledge (e.g. physicians and nurses)
- the development organisation (e.g. medical device expert and risk analysis supervisor)
- the researchers (e.g. process experts and technical experts from academia)

At this stage of the process no representatives from patients´ organisations was involved.

The target environment for the new medical device is an intensive care unit (ICU).

The development organisation has extensive experience in developing and maintaining medical devices, but not devices including software.

## 3.4 The Study of the Organisation

This section is divided in two parts, the first describing the preparations together with the organisation and the second the performed risk meetings were the data collection was made.

**Preparations with the Organisation.** The organisation had a defined risk management process for development of hardware but needed a risk management process adapted to software development. The research started out with an introductory discussion about the development process including the risk management process. It was clear that the study should focus on the two first steps in the risk management process, i.e. risk identification and risk analysis. As a result of these introductory discussions a process for risk identification and risk analysis designed for software systems was defined. The software risk process is described in Chapter 4.

**Data Collection.** Five risk meetings were held during the autumn of 2010, first meeting in September and the last meeting in December. The meetings lasted for approximately three hours each time. At least two representatives from each group of participants (users, development organisation, process experts and technical experts) were present at the meetings. During the meetings a risk was both identified and assessed at the same meeting. In total 152 risks were identified and 12 of them where given a high risk value. The software in the medical device has taken approximately 18-man month to develop at this stage of the risk process meetings.

The authors made active observations during the risk process meetings. To capture interesting aspects and pros and cons regarding the process the authors asked direct questions during the meetings. For example, if something was vague regarding the process or the product the authors asked for clarification.

All the authors have documented their observations during the risk process meetings, and the notes contain both direct observations and the authors' own reflections.

### 3.5 Analysis

The analysis the fourth step in the case study process (Figure 2) is based on the notes taken by the authors at the risk process meetings. After the meetings, the notes were compiled as a list of statements in a protocol and distributed among the authors. The group of authors compared and discussed interesting observations and reflections in close connection to the risk process meetings. The listed statements in the protocol were sorted into two different groups, with either "objective" observations or personal reflections. In the next analysis step the two groups of statements were sorted according to in which step of the risk analysis process they were noted. After that the observations and reflections were categorised according to the problems they affect. The purpose of this strategy was to get a better understanding of the material and make it easier to navigate.

During the on-going study a case study protocol was maintained containing purpose, procedures and detailed description of the course of events during the process. The case study protocol was updated over time.

All the material, including the results, was compiled into a preliminary technical report and includes the same information about the case study and the result as in this paper. The report was sent to representatives from the development organisation as part of the feedback process. The feedback process gives an opportunity to make clarifications from all parts and it also gives the authors conformation that the studied process is correctly. The representatives from the development organisation confirmed that the content of the technical report was consistent with theirs comprehension of the process and only minor details had to be corrected.

### 3.6 Validity

Validity of this kind of study can for example be analyzed with respect to construct validity, internal validity, external validity, and reliability [13].

*Construct validity* reflects to what extent the factors that are studied really represent what the researcher have in mind and what is investigated according to the research questions. In this study there were several different roles with different types of expertise involved. This could be a potential threat since there is always a risk of misunderstandings. One aspect that lowered this threat is that both the technical experts and the process experts had a long tradition of working together with the medical experts, which means that they had good knowledge of the investigated product and the usage of it. However, the risk cannot be ruled out totally.

It can also be noted that if, for example, medical terms were misunderstood by the researchers or the process experts, this would probably be a larger problem for the result of the conducted risk analysis than for the research results presented in this paper. The research was conducted as part of the risk analysis attempt and not seen as a something completely different by the participants. There was, of course, a wish to do an as good risk analysis as possible, which we also think is good for the research results.

*Internal validity* is important in studies of causal relationships. We have not identified any significant relations of this kind.

*External validity* is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to people outside the investigated case. The study was conducted with a limited set of participants from one single project. This means, of course, that the results cannot automatically be generalized to other organizations and projects. Instead it must be up to the reader to judge if it is reasonable to believe that the results are relevant also for another organization or project. However it should be noted that this is the typical situation in a case study. The case is studied in detail in order to learn as much as possible from it.

*Reliability* is concerned with to what extent the data and the analysis are dependent on the specific researchers. The reliability was addressed by conducting both the data collection and the data analysis as a group of researchers instead of one single researcher. The preliminary results were also sent to the other participants in the form of a technical report. This made it possible for the other participants to find possible error by the researchers.

## 4 Software Risk Process

In the risk identification process, different techniques can be used, such as checklist-based identification, development of prototypes, cost-benefit analysis, and scenario-based analysis [16]. In this case study, scenarios were chosen by the development organisation to be the main risk identification source. A scenario was defined as a chain of events, with a cause-effect relationship that describes a realistic diagnosis sequence during normal use, see Figure 3. Each scenario can be traced back to at least one requirement for the product. The scenarios cover both normal operation and special circumstances.

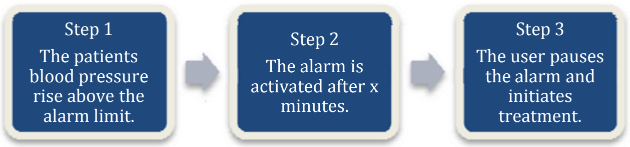
Step 1
The patients blood pressure rise above the alarm limit.
Step 2
The alarm is activated after x minutes.
Step 3
The user pauses the alarm and initiates treatment.

**Fig. 3.** Example scenario

Scenarios based on the requirements specification were used in the risk identification process as input. The design of the software risk process is shown in Figure 4.

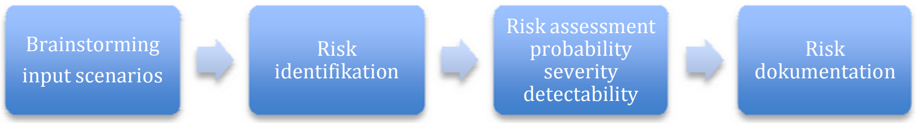
Brainstorming input scenarios
Risk identifikation
Risk assessment probability severity detectability
Risk dokumentation

**Fig. 4.** Software risk process

The risks were identified through brainstorming on each scenario were all participants suggested possible risks connected to the specific scenario. The medical device expert acted as facilitator during the brainstorming sessions. Then each risk was assessed separately according to probability, severity and detectability. Scales predefined by the Swedish national board of health and welfare was used for probability and severity assessment and all identified risks were documented during the meetings. Both scales are 4 graded scales. 4 on the probability scale correspond to "fault that will occur each month or more frequently at normal use" and 1 "fault will never occur or very unlikely. On the severity scale 4 will correspond to death or sever injury and 1 to discomfort or minor injury. The risk value, $R$, was calculated for each risk by multiplying the probability, $P$, by the given figure for severity, $S$, i.e., $R = P \times S$. The highest risk value a risk in this study can have is $R = 4 \times 4 = 16$. Detectability was assessed according to the three following statements "if the fault (hazard) **always** could be detected before a severe situation occurred", "if the fault (hazard) **sometimes** could be detected" or "if the fault (hazard) **never** could be detected". No figures were assigned to detectability so it is not a part of the calculated risk value R. The assessment of detectability was made after risk value was stated.

## 5 Results

In this section we present our results from observations made during the risk analysis of the blood pressure monitoring system described in Section 3. The results are grouped, with regards to the research questions, into three categories: system definition (RQ3), risk identification (RQ1) and risk assessment (RQ2). See Table 1, 2 and 3 to get a brief summary of the results.

## 5.1 System Definition

The whole system, or the product, is defined as consisting of the devices, including hardware and software, described in Section 3.3. In an effort to narrow down the scope of the risk analysis and focus on the in-house developed software the analyzed system was defined as consisting solely of the bedside monitor. Focus was on software functions and user interaction.

**Table 1.** Summary of the results concerning the system definition

| Area | Summary |
|---|---|
| System Boundary | The team had difficulties to decide whether a risk belonged to the system or the environment. |
| System Boundary | The team had to make assumptions about input from external devices and their reliability. |
| System Context | The target environment was not defined in detail and information about workload, user experience and physical layout of the target environment had to be supplied on the fly. |

The team had to make assumptions about the devices not included in the analyzed system, such as the patient monitor and the pressure sensors.

Major interfaces between the analyzed system and its environment were identified, such as the graphical user interface and some of the technical interfaces between the components in the whole system.

The target environment and the intended users were defined as the ICU respectively nurses and physicians at the ICU. Factors in the environment such as physical and mental working conditions, current practice and rules, were described when questions about them arose. This also applied to differences between the different user categories.

It was difficult for some risks, with the chosen system boundaries, to decide if they were to be considered or not. This was especially true for risks that arose from bad data from the measuring equipment or from incorrectly connected devices, i.e. should sensor failures that result in bad readings, which is then used by the bedside monitor, be analyzed.

## 5.2 Risk Identification

The used process puts emphasis on the users and the users' interaction with the system. In the process this is achieved by using realistic scenarios based on knowledge of the target environment.

The risk identification was performed by going through each step of the scenarios, and for each step discuss if there were any risks associated with that step. During this part of the process the user representatives dominated the discussions. They had more background information about the scenarios, e.g. medical knowledge, and the target environment e.g. working conditions and were thus better suited to identify risks.

**Table 2.** Summary of the results concerning risk identification

| Area | Summary |
|---|---|
| Scenario | The user representatives had better background knowledge than development representatives. |
| Scenario | It was unclear if risk identification in a given event should be done independently of the scenario or if the identification should be constrained by the scenario history. |
| Scenario | Technical risks were not restricted to the scenario they were identified in, as opposed to user risks, which were to a larger degree only valid in a specific scenario context. |
| Scenario | The design of the scenarios clearly impacted the outcome of the risk identification. |

The representatives from the development organization shared valuable insights about the technical nature of the system. In particular their expert knowledge of the software and the graphical user interface was of value to the team. They had in general less influence on the discussion than the user representatives.

Different views on how previous steps in a scenario should affect the current step were noted during the risk assessment meetings. Some argued that previous steps should put a constraint on the current step whereas others argued that the current step should be analysed independent of previous steps, e.g. using the scenario in Figure 3 should risks be identified when pausing the alarm in general or only when a patient has a high blood pressure during x minutes?

There was a tendency that a perceived probability of a particular risk or the severity of its consequences influenced the risk identification. At times the team argued that a risk that was not very probable or had very mild consequences should not be considered to be a risk.

In some of the cases where an identified risk originated from incomplete requirements, this was treated as a fault in the requirements specification.

Although the scenario-based method focuses on user-interaction and user related risks, some technical risks were found, mostly concerning interfaces. The technical risks have in common that they are more general in nature than the user related risks, and they are not bound to a specific scenario.

It became evident that the scenario composition had an impact on the outcome of the risk identification; a wrongly constructed scenario would catch unrealistic behaviour. In some cases scenarios had to be adjusted because they did not describe the system or user behaviour well.

The method was expected to be able to identify all major user risks and provide full coverage of relevant features by using at least one scenario for each relevant product requirement. Critical functionality, such as the automatic alarm function, was analysed during several scenarios.

### 5.3 Risk Assessment

The risk assessment was conducted using a method influenced from the development organisation's prior risk management process for hardware projects. The method specifies three variables, severity, probability and detectability, that are to be assessed based on normal usage of the system.

**Table 3.** Summary of the results concerning risk assessment

| Area | Summary |
|---|---|
| System Context | The risk assessment was made under the assumption of normal use, which was defined as the average workload during a year. |
| Scenario | The user representatives, due to their extensive medical domain knowledge, dominated the estimation of severity and, to a lesser extent, probability. |
| Detectability | The team had problems with estimating detectability and refrained from doing it for the majority of the identified risks. |
| Detectability | The estimation of severity, probability and detectability was sometimes influenced by the other values, e.g. a low probability would result in a low severity; if a risk is detectable then it is not likely to happen. |
| System Boundary | The chosen system boundary was seen as too narrow because it did not include all devices in the final product. A risk could have catastrophic consequences in the chosen system but when considering the whole product the risk would be non-existing or less severe. |

Prior to the risk assessment the team had to define what normal usage meant for the actual system. It was defined as the average workload, e.g. the average number of patients at the ICU and the average duration a patient is connected to the system. The process does not give any help on how normal use is to be defined.

Each risk was assessed separately from the other risks, starting with severity followed by probability and finally detectability.

During the severity assessment the user representatives had great impact on the results. Typically, they would be the only persons able to determine the consequence of a particular risk in the target environment.

Assessing a risk's probability required both the users and the developers. Risks associated with user-interaction had probabilities assigned based on the current situation at the ICU and on previous experience with similar systems. Technical risks had their probabilities assigned based on the opinion of the developers. The team did not assign probabilities to pure software related risks.

The team had problems with assessing the detectability value of a risk. In some cases it was difficult or even impossible to assign an appropriate value, which was usually the case when the risk had something to do with being unaware of an event. Usually the users were the only participants that could determine if a risk was detectable. Due to the difficulties of estimating the detectability the team refrained from estimating the value for most of the risks.

An issue that had to be solved during the risk assessment meetings was that it was unclear what the severity and probability values actually meant and how they were related to each other. It was determined from discussions that the severity is the worst-case consequence of a risk and the probability is how often risk occurs, independent of its consequences.

Although, the assessment of the risk values should be independent of each other it was in some cases hard to separate the discussion about severity and probability. When detectability could be assigned it sometimes influenced the assessment of probability and severity, i.e. some argued that if a problem was detected actions would be taken to prevent it form happening or leading to an accident.

Another issue was that the system definition was impractical when assessing some risks, it was seen as too narrow. To alleviate this, the full system definition was used. In the narrow definition a risk could be considered to have catastrophic consequences but in the wider definition the risk was prevented or mitigated by devices not included in the narrow system definition, e.g. the patient monitor.

## 6 Discussion and Conclusion

In this section we discuss our results and present the conclusions we made from our analysis. The discussion is organized, with the aim of addressing the research questions, into four areas: system boundary (RQ3), system context (RQ1 and RQ2), scenario (RQ1) and estimation (RQ2). Each of these areas address problems that we found especially challenging during the risk analysis which can be considered when a new risk management process is defined.

### 6.1 System Boundary

In systems theory safety is an emergent property on the system level [17]. Even so, one of the purposes of the study was to see if it was possible to do risk analysis on a smaller part of the whole system, e.g. the in-house developed software and patient monitoring device.

From our results we can draw the conclusion that the system boundaries must be set carefully and not without considering dependencies between components.

As observed in the risk identification step it was necessary to make assumptions about signals from external devices used in the analysed system. Later in the risk assessment step the existence of these external devices could be used to argue that certain risks was non-existing or had low severity.

Before defining the system boundary it should be clear how components are coupled. Components with weak coupling might be analysed independently and components with strong coupling should be analysed together.

## 6.2 System Context

The system context like users, physical and psychological work conditions affect the identification and assessment of risks. It is therefore important that the system context is defined during the analysis.

In the studied risk process normal use is used as an indicator on how the system will be used in the target environment. Normal use is defined as an average of the workload on the system in the target environment. This is a simple approach but it gives no detailed understanding about how the system is used and how it affects the risk analysis.

By describing normal use in a more quantitative manner e.g. using a scenario or use case, a more nuanced picture can be made about the use of the system in its context. The description should not only describe for how long and how often the system is used but also where and when. The description could be augmented with special case scenarios where high load and low load could be defined.

## 6.3 Scenarios

The studied scenario-based risk identification method focuses on user interaction and user related risks. Some technical risks were also identified using the scenarios and the nature of these risks relates primarily to user friendliness that displayed values are correct. Since technical risks are of a more general nature and not scenario-specific there is a need for a separate risk identification regarding these risks, preferably performed by the software development team that possesses domain knowledge of the system. There is also a need for risk identification of external factors for example process and project risks.

The scenarios have to be designed in a way so they reflect the system functions in as correctly as possible. It is not possible to determine that a scenario is incorrect based on the assumption that the course of events is unlikely. The balance between plausible scenarios and special cases has to be considered. When the scenarios are designed, a possible way could be to let the users and developers work separately. After the separate design process the scenarios could then be discussed and decided on in a plenary discussion before the risk identification starts.

The scenarios used in this case have no contextual description attached to the scenario. It could be of value to put a scenario in its context and describe the presumptions made regarding the scenario for example describe the working situation, if it is an “ordinary” day with acceptable numbers of patient or a very stressful day with a lot patients with severe traumas.

When the scenarios were discussed step by step, it could be noted that the user representatives, as expected, are the dominant part, since they possess domain knowledge regarding the target environment and medical issues. The developers had a more peripheral role and were consulted regarding technical aspects of the system.

A possible solution to the dominance factor could be to have very strict control of the meetings, with the ambition to get the opinion from all the participants, for example give specific time slots to each participant.

## 6.4 Estimation

The qualitative nature of estimating the value of the risk quantities, in particular that it is based on the participant's subjective opinions makes the result quite uncertain. It is important to define and separate the different estimations to be made and strictly comply the predefined scales.

Detectability was not estimated for the majority of the risks due to several reasons. The scale was considered imprecise and did not assist the participants in the estimation effort, as the scales for severity and probability did. Another problem was that the concept of detectability was not well understood.

The used scale defines three levels of detectability: a risk is, *never, sometimes* or *always* detected. It was found that these words lack precision and are subject to personal interpretation. For instance, does always mean that a risk is always detected, most of the time or only when it can be observed? The scale gives a false impression that detectability can be measured quantitatively although it is a qualitative property. Instead of detectability we would suggest that it is better to use observability – if a risk is observable then it can be detected and vice versa – using the scale: a risk is *direct observable, indirect observable, unobservable*.

In addition to the observed problems it could be argued that detectability should be considered as a mitigating factor and be estimated during the risk treatment step. There exists at least two counter-arguments for this: first, the expert knowledge that is required to determine the detectability might not be available when risk treatment is performed; secondly, the detectability value would give additional information when prioritizing risks for further analysis and treatment.

After completing the risk assessment the development organization decided, based on the encountered problems, to remove detectability from the process. Although this simplifies the process it removes potentially important information about risks. There is a need for further research on how to define and estimate detectability of identified risks.

## 6.5 Further Research

In this paper we present intermediary results from an ongoing case study. The results are from the first two phases of the studied risk management process: risk identification and risk analysis. We are currently monitoring the follow up meetings of the risk management were risk mitigation and residual risk analysis is performed. The use, definition and estimation of detectability is further discussed and analysed.

We intend to use our results to design an improved version of the risk management process. Our focus will be on scenario based risk analysis in an iterative development process, and the link between the risk analysis and the overall development process.

**Acknowledgments.** The authors would like to gratefully acknowledge the persons involved in this case study. The authors would also like to acknowledge Gyllenstiernska Krapperup-stiftelsen for funding the research studies of Christin Lindholm.This work was also partly funded by the The Swedish Foundation for Strategic Research under a grant to Lund University for ENGROSS - ENabling GROwing Software Systems. The work has also been partially supported by VINNOVA.

## References

1. Gall, H.: Functional Safety IEC 61508 / IEC 61511 The Impact to Certification and the User. In: IEEE International Conference on Computer Systems and Applications (2008)
2. Lindholm, C., Notander, J.P., Höst, M.: Software Risk Analysis in Medical Device Development. Accepted for Publication in Proceedings of EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Oulu, Finland, August 30-September 2 (2011)
3. Garde, S., Knaup, P.: Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology. Requirements Engineering, 265–278 (2006)
4. Fairley, R.E.: Software Risk Management. IEEE Software, 101 (May/June 2005)
5. Rakitin, S.R.: Coping with Defective Software in Medical Devices. IEEE Computer 39(4), 40–45 (2006)
6. Commission of the European Communities, Council Directive 93/42/EEC EEC concerning medical devices (1993)
7. U.S. Food and Drug Administration, Federal Food, Drug and Cosmetic Act section 201(h) (2005)
8. Hall, E.M.: Managing Risk: Methods for Software systems development. Addison Wesley (1998)
9. McCaffery, F., Burton, J., Richardson, I.: Risk management capability model for the development of medical device software. Software Quality Journal (18), 81–107 (2010)
10. McCaffery, F., Burton, J., Richardson, I.: Improving Software Risk Management in a Medical Device Company. In: Proceedings of International Conference on Software Engineering (ICSE), Vancouver Canada (2009)
11. Schmuland, C.: Value- Added Medical-Device Risk Management. IEEE Transactions on Device and Materials Reliability 5(3), 488–493 (2005)
12. Sayre, K., Kenner, J., Jones, P.: Safety Models: An Analytical Tool for Risk Analysis of Medical Device Systems. In: Proceedings of 14th IEEE Symposium on Computer-Based Medical Systems (CMBS 2001), Maryland, USA (2001)
13. Yin, R.K.: Case Study Research Design and Methods, 3rd edn. Sage, Thousand Oaks (2003)
14. Robson, C.: Real World Research, 2nd edn. Blackwell Publishers Ltd., Oxford (2002)
15. Runeson, P., Höst, M.: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering 14(2), 131–164 (2009)
16. Boehm, B.: Software Risk Management: Principles and Practices. IEEE Software 8(1), 32–41 (1991)
17. Leveson, N.G.: Safeware: System Safety and Computers, Addison-Wesley Professional (1995)

# Integrating Manual and Automatic Risk Assessment for Risk-Based Testing

Michael Felderer[1], Christian Haisjackl[1], Ruth Breu[1], and Johannes Motz[2]

[1] Institute of Computer Science, University of Innsbruck, Austria
{michael.felderer,christian.haisjackl,ruth.breu}@uibk.ac.at
[2] Kapsch CarrierCom AG, Vienna, Austria
johannes.motz@kapsch.net

**Abstract.** In this paper we define a model-based risk assessment procedure that integrates automatic risk assessment by static analysis, semi-automatic risk assessment and guided manual risk assessment. In this process probability and impact criteria are determined by metrics which are combined to estimate the risk of specific system development artifacts. The risk values are propagated to the assigned test cases providing a prioritization of test cases. This supports to optimize the allocation of limited testing time and budget in a risk-based testing methodology. Therefore, we embed our risk assessment process into a generic risk-based testing methodology. The calculation of probability and impact metrics is based on system and requirements artifacts which are formalized as model elements. Additional time metrics consider the temporal development of the system under test and take for instance the bug and version history of the system into account. The risk assessment procedure integrates several stakeholders and is explained by a running example.

## 1 Introduction

In many application domains, testing has to be done under severe pressure due to limited resources with the consequence that only a subset of all relevant test cases can be executed. In this context, risk-based testing approaches are more and more applied to prioritize test cases based on risks. A risk is the chance of injury, damage or loss and typically determined by the probability of its occurrence and its impact [1]. A core activity in every risk-based testing process is the risk assessment because it determines the significance of the risk values assigned to tests and therefore the quality of the overall risk-based testing process.

In current practice, mainly human experts conduct a risk assessment. Therefore the process of risk assessment is expensive, time consuming, and contains nondeterminism regarding subjective human decisions. This makes the re-assessment of risks during the testing process a challenging task. To avoid risk assessment to become a critical cost driver in software development projects, we propose an approach to automate parts of the risk assessment. The basis for this partial automation is the fragmentation of its probability and impact into several criteria. Each criterion in the resulting set of probability and impact criteria has an attached metrics which is computed automatically, semi-automatically or manually.

To aggregate all probability and impact criteria to one risk value and to assign this value to tests we attach the risk to so called features. Features specify capabilities of a component and link tests, requirements, and system components together. Risk coefficients are computed for features and propagated to the assigned tests. The probability and impact criteria underlying the risk coefficient are determined based on the model elements assigned to the feature.

Differing from other approaches we additionally consider separate time criteria such as the version history or the test history depending on the temporal development of the project. Time criteria can be computed automatically and are considered as ratios that reduce the probability.

The risk assessment should start early in a software project to provide decision support. Due to changes of the requirements or the design, risks have to be reassessed regularly. Additionally not all artifacts are available in all phases of a project and therefore it may not be possible to automatically compute specific metrics at a specific point in time. Therefore the evaluation type and the evaluation procedure of specific metrics may change during the life cycle of a project. For example a metrics for complexity may be estimated manually at the beginning of a project and automatically in later project phases.

In our approach we focus on product risks where the primary effect of a potential problem is on the quality of the product itself. Since testing is about finding problems in products, risk-based testing is about product risks [2]. Thus, we do not consider project risks related to management and control of the test project like the capabilities of the staff or strict deadlines. The focus on product risks is additionally motivated by the fact that product risks directly influence the test plan and that product risks are typically more objective because they do not consider the capabilities of the project staff. If desired by the software project management, project risks can be considered in our approach as additional criteria [3].

In this paper we contribute to the integration of risk assessment into risk-based testing in several ways. First we introduce a generic risk-based testing process that highlights risk assessment as a core activity. We then define a static risk assessment model that considers a probability, impact, and time factor which have not been considered separately in other approaches. Moreover, each factor is determined by several criteria. Finally, we define a risk assessment procedure based on the risk assessment model and give an example. In the risk assessment procedure each factor is calculated by an aggregation function based on several criteria. Each criterion is determined by a metrics that is evaluated manually, semi-automatically or automatically and that can change over time.

This paper is structured as follows. In Section 2, we present related work, and in Section 3 we define a generic risk-based testing process. In Section 4, we then define the underlying risk assessment model, and in Section 5 we discuss our risk assessment procedure. Finally, in Section 6, we draw conclusions.

## 2 Related Work

The importance of risk management for software engineering [4] has been addressed in several risk management approaches tailored to software engineering processes. Already the spiral model of Boehm [5] explicitly includes risk management within software development. For instance, the Riskit method [6] provides a sound theoretical foundation of risk management with a focus on the qualitative understanding of risks before their possible quantification. Furthermore, the Riskit method provides a defined process for conducting risk management which is supported by various techniques and guidelines. Ropponen [3] highlights the importance of mature project management for risk management.

Karolak [7] proposes a risk management process for software engineering that contains the activities risk identification, risk strategy, risk assessment, risk mitigation, and risk prediction.

Risk-based testing is a test-based approach to risk management. Amland [8] proposes a risk-based testing approach that is based on Karolak's risk management process comprising the following steps and the corresponding risk management activities: planning (risk identification and risk strategy), identification of risk indicators (part of risk assessment), identification of the cost of a failure (part of risk assessment), identification of critical elements (part of risk assessment), test execution (risk mitigation), and estimation of completion (risk reporting and risk prediction).

Bach [9] presents a pragmatic approach to risk-based testing grounded on a heuristic software risk analysis. Bach distinguishes inside-out risk analysis starting with details about a situation and identifying associated risk, and outside-in risk analysis starting with a set of potential risks and matching them to the details of the situation.

Taxonomy–based risk identification, e.g., as proposed by the SEI [10], supports the risk analysis.

Risk-based testing techniques have rarely been applied on the model level. RiteDAP [11] is a model-based approach to risk-based system testing that uses annotated UML activity diagrams for test case generation and prioritization. This approach does not consider how the risk values are determined and only considers the risk-based generation of test cases. But risk-based testing techniques have so far not been integrated into a model-driven system and test evolution process, and have not been applied for model-based regression testing.

Stallbaum and Metzger [12] introduce a model-driven risk-based testing approach similar to our methodology. But the approach of Stallbaum and Metzger does not consider the system model and time criteria.

The basis for risk-based testing on the model level is model-driven risk analysis. The CORAS method [13] addresses this task and defines a metamodel for risk assessment. But CORAS is a defensive approach restricted to assets that already exist. Additionally, risk-based testing based on CORAS has not been conducted so far. There are several other model-driven risk analysis approaches like fault trees [14], attack trees [15], misuse cases [16] or Tropos Goal-Risk modeling [17] that are useful for risk-based testing.

McCall [18] distinguishes factors which describe the external view of the software, as viewed by the users, criteria which describe the internal view of the software as seen by the developers and metrics which provide a scale and a method for measurement.

Many researchers have addressed the problem of risk assessment using guide lines, checklists, heuristics, taxonomies, and risk criteria [19]. All these risk assessment approaches strongly rely on experts that perform the assessment. Most approaches that enable automation in risk assessment employ metrics based on code artifacts.

There are several approaches, e.g., [20,21] that employ change metrics like the change rate, code metrics like the number of code lines and complexity metrics like the McCabe complexity [22] to predict the failure rate. Experiments indicate that the combination of code and design metrics has a higher performance for predicting the error-proneness of modules than just code or design metrics [23]. These approaches are not directly related to risk-based testing as our approach but they form the basis for determining the probability of risks.

For instance, Nagappan et al. [20] investigate the correlation between classical function, class, and module metrics and failures for selected projects. The authors state that for each project, a set of metrics that correlates with the failure rate exists but that there is no single set of metrics that fits for all projects. This result is very useful for the assessment of the probability of a risk in our approach as it argues for a mix of manual and automatic risk determination.

Illes-Seifert and Paech [21] explore the relationship of a file's history and its fault-proneness, and show that a software's history is a good indicator for its quality. There is not one indicator for quality but the number of changes, the number of distinct authors, as well as the the file's age are good indicators for a file's defect count in all projects. Illes-Seifert and Paech do not consider risk assessment. But the results are a good basis for the determination of time criteria in our approach.

Additionally, also vulnerability databases, e.g., the national vulnerability database [24] or the open source vulnerability database [25] are powerful sources for risk estimation [26].

Similar to our approach the Common Vulnerability Scoring Standard [27] uses a variety of metrics to calculate a risk coefficient. The metrics are organized in three different groups: one basic and two optional. Although the approach focuses on security risks, its flexibility can also be applied to our approach for integrating optional metrics.

## 3 Generic Risk-Based Testing Process

Risk-based Testing (RBT) is a type of software testing that considers risks assigned to test items for designing, evaluating, and analyzing tests [8,9]. A risk is the chance of injury, damage or loss. A risk is therefore something that might happen and has a potential loss or impact associated with that event.

Typically, in risk-based testing the standard risk model is based on the two factors *probability* ($P$), determining the likelihood that a fault assigned to a risk occurs, and *impact* (consequence or cost) ($I$), determining the cost of the assigned fault if it occurs in production.

The probability typically considers technical criteria, e.g., complexity, and the impact considers business criteria such as monetary loss or reputation. These criteria can be determined by automatically, semi-automatically or manually computed metrics. Our risk computation model follows the approach of McCall [18] who suggests using a divide-and-conquer strategy involving a hierarchy of three types of characteristics: factors, criteria, and metrics. On a high level, the factors describe the external view of the software, as viewed by the users. Then, the criteria depict the internal view of the software, as seen by the developer. And the metrics are defined to provide a scale and method for measurement.

Mathematically, the risk (coefficient) $R$ of an artifact $a$ can be expressed by the probability $P$ and the impact $I$ as follows:

$$R(a) = P(a) \circ I(a)$$

The depicted operation $\circ$ is typically a multiplication of numbers or the cross product of two numbers or letters, but can principally be an arbitrary mathematical operation.

Risk-based testing is integrated into a testing process as shown in Fig. 1. Such a process integrates elements of risk management and test management. In most cases, risk-based testing is limited to risk prioritization and its full potential to improve software quality by mitigating risk and optimizing test resource allocation is not exploited. To improve this situation, we define a comprehensive risk-based testing methodology that is aligned with established risk-based testing processes such as Amland [8] or the ISTQB [28]. Our generic risk-based testing process therefore contains activities for the identification of risks, the test planning, the risk assessment, the test design and the evaluation.

In Fig. 1 the activities are depicted as ellipses, and the artifacts as input and output of the activities are depicted as rectangles. This paper focuses on the core activity of Risk Assessment which is therefore highlighted in Fig. 1.

Our generic risk-based testing process depicted in Fig. 1 consists of the phases *Risk Identification*, *Test Planning*, *Risk Assessment*, *Test Design*, and *Static and Dynamic Evaluation* explained in the subsequent paragraphs.

In the **Risk Identification** phase risk items are identified and stored in a list of risk items. Risk items are the elements under test for which the risk is calculated. Therefore risk items need to be concrete such that a risk calculation and an assignment of tests are possible. Risk items can be development artifacts, e.g., requirements or components but also different types of risks such as product, project, strategic or external risks. According to practical experiences [29], the maximum number of risk items should be limited.

In the **Test Planning** phase the test strategy is defined. The test strategy contains test prioritization criteria, test methods, exit criteria and the test effort under consideration of risk aspects. Although our approach is independent of a concrete test strategy, we assume that a given test strategy contains a basic risk classification scheme as depicted in Fig. 4. Such a classification scheme demonstrates how the calculated risk values can be further employed in our risk-based testing methodology.

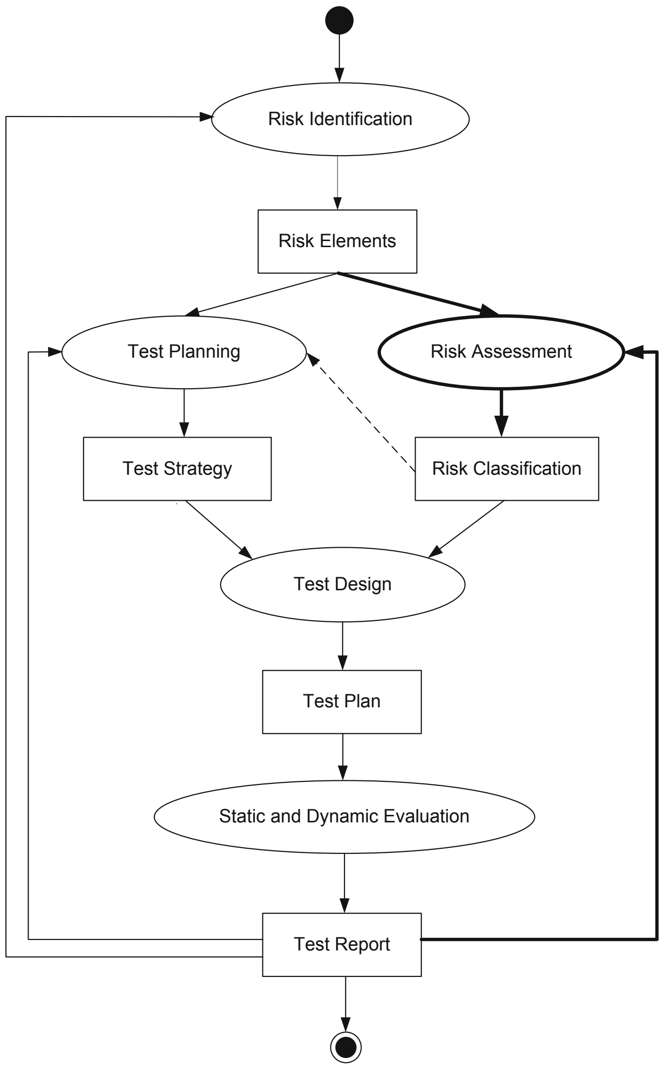

**Fig. 1.** Generic Risk-based Testing Process

In the **Risk Assessment** phase the risk coefficient is calculated for each risk item based on probability and impact factors. The probability $P$ and the impact $I$ are typically evaluated by several weighted criteria and can be computed as follows for a specific artifact $a$:

$$P(a) = \frac{\sum_{j=0}^{m} p_j \cdot w_j}{\sum_{j=0}^{m} w_j} \quad , \quad I(a) = \frac{\sum_{j=0}^{n} i_j \cdot w_j}{\sum_{j=0}^{n} w_j},$$

where $p_j$ are values for probability criteria, $i_j$ are values for impact criteria, and $w_j$ are weight values for the criteria. The range of $p$, $i$, and $w$ are typically natural or real numbers in a specific range, e.g., for $p$ and $i$ we suggest natural numbers between 0 and 9 (so the probability can be interpreted as percentage; we skipped the value 10 (suggesting 100%) because we assume that a component does not fail for sure), and for $w$ we suggest real numbers between 0 and 1 (so the weight can be naturally interpreted as scaling factor).

The values of the metrics are computed automatically, semi-automatically or manually based on an evaluation procedure, and the weights are set manually. The responsibility for setting non–automatically derived values respective checking the automatically derived values should be distributed among different stakeholders, e.g., project manager, software developer, software architect, test manager, and customer.

The impact $I$ is computed by analogy to the probability and typically evaluated manually by customers or product managers. Without loss of generality, we use the same range for impact values as for probability values. In our adapted risk assessment approach we additionally consider time criteria which take metrics into account that change over time because of the products life cycle.

A time factor is a ratio that reduces the probability value which is reflected in our adapted formula for determining the risk coefficient $R$ of an artifact $a$:

$$R(a) = (P(a) \cdot T(a)) \circ I(a),$$

where $T(a)$ is the time factor that is multiplied with the probability $P(a)$ and has a range between 0 and 1. The resulting product is then combined with the impact $I(a)$.

The time factor is computed by the mean of the values of k time criteria by the following formula:

$$T(a) = \frac{\sum_{j=0}^{k} t_j}{k},$$

where $t_j$ are values for time criteria. We assume that over time, stable parts of the software have a lower probability to fail, so we scale the probability in this way.

Based on the impact and probability factor, the classification of risk items into risk levels is done. A typical way to do so is the definition of a risk matrix as shown in Fig. 4. The risk classification is considered in the test planning phase. An example for the risk assessment procedure and the risk classification is presented in Section 5.

In the **Test Design** phase a concrete test plan is defined based on the test strategy and the risk classification. The test plan contains a list of test cases that has been designed under consideration of risk aspects defined in the test strategy, the risk classification and the given resources.

In the **Static and Dynamic Evaluation** phase the test plan is executed manually and/or automatically. Based on the execution result a test report is generated. The evaluation phase may contain dynamic evaluation, i.e., test execution and static evaluation, e.g., inspections of documents that are assigned to risk items that have been classified as risky.

The phases of risk identification, risk assessment and evaluation are together called the risk analysis phase. As shown in Fig. 1, the process of risk-based testing is iterative to consider changes of the risk items and previous evaluation results in the risk analysis.

If the RBT process defined above is integrated with an existing test and software development process, then the benefits of RBT may be as follows:

– Reduced resource consumption (i.e., more efficient testing) by saving time and money
– Mitigation of risks
– Improved quality by spending more time on critical functions and identifying them as early as possible
– Early identification of test issues, which can be started when the requirements specification is completed
– Support for the optimization of the software testing process by moving the test phase from the latter part of the project to the start and allows the concept of life cycle testing to be implemented
– Identifying critical areas of the application in the requirement stage. This assists in the test process but also in the development process because it supports the optimization of resources, e.g., time, budget, or person days.

## 4 Risk Assessment Model

In this section we present the risk assessment model which is the basis for the risk assessment procedure defined in the next section. The risk assessment model is shown in Fig. 2.

The model contains the package *Risk* which includes the core risk assessment elements and the packages *Requirements*, *Product*, *Implementation*, and *Test* which include requirements, system architecture, implementation, plus test artifacts relevant for risk-based testing. In the following subsections we explain each package in detail. The modeling of the package *Risk* is more granular than the others because it contains the core artifacts of the risk assessment procedure.

### 4.1 Requirements

The package *Requirements* contains the requirements each defining a certain functional or non-functional property of the system. A *Requirement* is implemented by a set of *Feature* elements and is related to a set of *Document* elements.

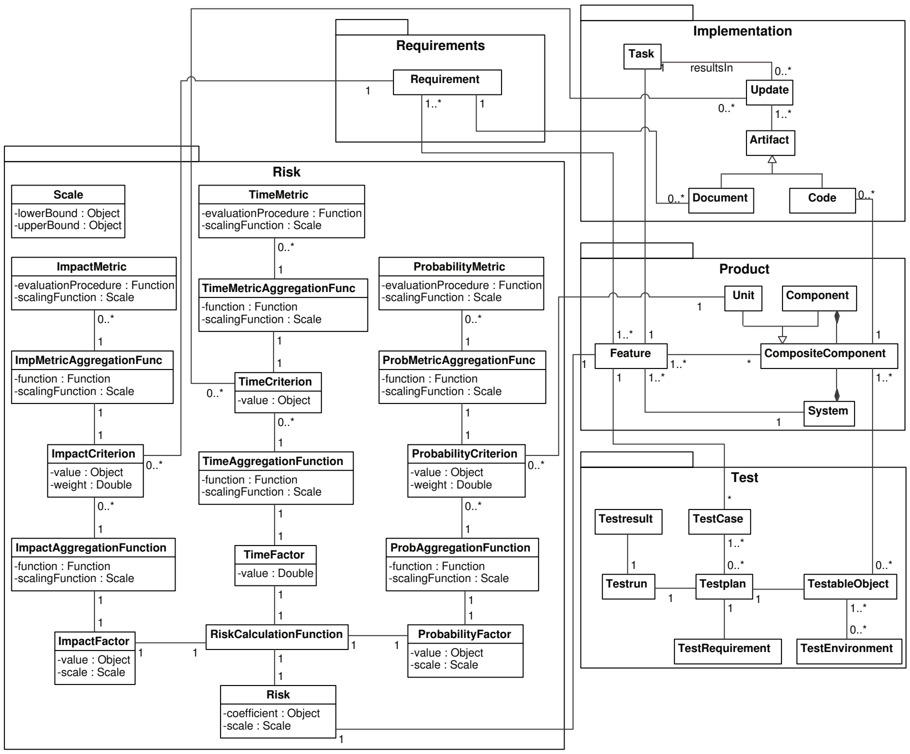

**Fig. 2.** Model Elements for Risk Assessment

## 4.2 Product

The package *Product* contains the core elements of the system. A *Feature* describes one specific capability of a system or component. It is the central concept to plan and control the implementation. *Feature*s are the core model elements which link all packages together. A *Feature* validates *Requirement*s and is verified by *TestCase*s. Therefore *Feature*s are the natural elements for the assignment of *Risk*s. Each *Feature* is implemented by several *Task*s and assigned to a *System* and several *CompositeComponent*s.

A *System* consists of several *CompositeComponent*s which are either *Component*s or *Unit*s. A *CompositeComponent* is an installable artifact that provides the functionality of several *Feature*s and is defined in a hierarchy of *Component*s and *Unit*s which are the leaves of the hierarchy. Because all modern software products are structured in a hierarchical way, the proposed structure can be applied for most software products. For instance, the components of a software architecture in the modeling language UML or the package structure in the programming language Java are hierarchical and form a hierarchy of *Component*s and *Unit*s.

## 4.3 Implementation

The package *Implementation* contains the concrete implementation artifacts *Code* and *Document*. *Code* represents implemented or generated source code, and *Document* represents a specification or documentation file, e.g., text files with user requirements or UML diagrams. If an *Artifact* is changed, an *Update* is created containing the new version. A *Task* groups several *Update*s and addresses the implementation or adaptation of a *Feature*.

## 4.4 Test

Several *TestCase*s form a *Testplan* which has assigned *TestRequirement* elements defining constraints on the test plan such as coverage criteria. The execution of a *Testplan* is a *Testrun* which has several *Testresult* elements assigned. A *Testplan* is executed on a *TestableObject* which is embedded in a *TestEnvironment* and related to a *CompositeComponent*.

## 4.5 Risk

The *Risk* is computed as defined in Section 3 based on the *ImpactFactor*, the *TimeFactor* and the *ProbabilityFactor* which are combined by a *RiskCalculationFunction*. The *Risk*, *ProbabilityFactor* and *ImpactFactor* have a value and a scale. The *ImpactFactor* is computed by an *ImpactAggregationFunction* based on *ImpactCriteria*, the *ProbabilityFactor* by a *ProbabilityAggregationFunction* based on *ProbabilityCriteria*, and the *TimeFactor* by a *TimeAggregationFunction* based on *TimeCriteria*. In this paper we apply the function mean as aggregation function.

Each *ProbabilityCriterion*, *TimeCriterion* and *ImpactCriterion* has a metric aggregation function of type *ProbabilityMetricAggregationFunction*, *TimeMetricAggregationFunction* or *ImpactMetricAggregationFunction*. The metric aggregation function contains all information to calculate the values of the criteria. Each aggregation function is based on metrics, i.e., *ImpactMetric*, *TimeMetric*, or *ProbabilityMetric*. Each metric has an evaluation procedure which can be of type manual, semi-automatic, or automatic, and a scaling function to normalize values. To make this paper more readable, we often identify metrics and the computed values.

The *Risk*, *ImpactFactor*, and *ProbabilityFactor* have a value and a scale for that value (for the *Risk* the value is called the coefficient). Each *ImpactCriterion* and *ProbabilityCriterion* has an additional weight value defining a ratio that a specific criterion has for the overall risk computation. The *TimeFactor* has in contrast to the other two factors only a value and no scale, because we assume that the time factor is a rational number between 0 and 1. Furthermore, the *TimeCriteria* do not have an additional weight because they are per definition reduction factors for probability criteria.

For evaluating the risk coefficient efficiently, the different factors are assigned to specific artifacts in our model.

- **Feature:** A *Feature* describes a specific functionality of the whole system. The test cases are attached to features. Therefore, the risk coefficient is assigned to this artifact.
- **Unit:** A *Unit* is the technical implementation of a *Feature*. Failures evolve from *Unit*s and the probability that failures occur influences the value of the probability factor and therefore also the risk coefficient. Thus, the *ProbabilityCriteria* are assigned to the *Unit*s.
- **Requirements:** The *Requirement*s represent the view of the customer on a system. If a *Feature* fails, a *Requirement* cannot be fulfilled and damage (monetary or ideally) is caused. Thus, the *ImpactCriteria* are assigned to the *Requirement*s.
- **Components:** The *Component*s consist of several *Units* and form subsystems, where the same directives can be applied. Thus, we attached the weights to the *Components*, so that the same weights can be applied to the associated Units.

Each *Feature* belongs to one *Component*, implements one or more *Requirement*s and uses one or more *Unit*s. Exactly these relationships are used for calculating the risk coefficient. The aggregated values, e.g., by the aggregation function mean (which we consider as aggregation function in the remainder of this document), of the *ImpactCriteria* from the *Requirement*s and the mean values of the single *ProbabilityCriteria* of the *Unit*s are weighted by the *Component*'s weights. Afterwards, both resulting factors are set into relation (by multiplication, cross product, etc.) for calculating the final *Risk*. The main principle behind our evaluation procedure is the separation of the evaluation of the impact and the probability factor.

The evaluation of the probability factor is based on internal system artifacts, i.e., units in our respect, and the evaluation of the impact is based on external requirements artifacts, i.e., user requirements in our respect. Therefore our risk assessment model can be adapted to other development and testing processes because most practical processes have a separation between internal and external artifacts.

As mentioned above, the *ProbabilityFactor* and *ImpactFactor* are based on several criteria, e.g., code complexity, functional complexity, concurrency, or testability for *ProbabilityCriteria*, and, e.g., availability, usage, importance, image loss, or monetary loss for the *ImpactCriteria*. Each of these criteria has its own weight at the level of *Component*s. If a *Feature* has more *Unit*s or *Requirement*s, the mean value of a specific criterion is calculated afterwards multiplied by the *Component*'s weight. As next step, the mean of these values is generated as final factor for the risk coefficient calculation.

So far we have only considered probability and impact factors but we have not considered their dynamics and the temporal development of *Risk* values. The *Risk* changes only if a *Requirement* is redefined or a *Unit* is adapted. As a *Feature* has a life cycle, the same holds for a *Risk*. But so far we have only considered criteria that are determined statically. To address this problem, we introduced an additional factor, called *TimeFactor*. The criteria of the *TimeFactor* are adapted dynamically and are, e.g., change history, bug reports, test history or software maturity, which change its value during the life cycle of a *Feature*. We have integrated the time factor as a scaling factor between 0 and 1, that adapts the *ProbabilityFactor*. Initially, the *TimeFactor* has the value 1 which means, that the probability is not lowered. If a specific *Feature* turns out to be more stable than other *Feature*s, some *TimeCritera* will be lowered and reduce the probability of that *Feature*.

In the remainder of this document, we often use the singular or plural form of the name of a metamodel element to refer to instances of it.

## 5 Risk Assessment Procedure

In this section we explain the algorithm for determining the risk coefficient by an abstract example. The example refers to the activity *Risk Assessment* of the generic risk-based testing process shown in Figure 1. The risk elements are *Feature* elements and the resulting risk classification is based on the computed risk coefficients. As the focus of this paper is the risk assessment procedure itself, we do not explicitly consider adequate test reporting techniques.

The example of this paper is based on three *ProbabilityCriteria*, i.e., code metrics factor, functional complexity and testability, two *ImpactCriteria*, i.e., usage and importance, and two *TimeCriteria*, i.e., change history and test history. Without loss of generality, the values of probability and impact criteria have an integer range between 0 and 9. The values of time criteria have a real number between 0 and 1. Additionally, each criterion except the time criteria have a weight. We consider the weight as a real number between 0 to 1. As aggregation function we apply the function mean. All values are rounded to one decimal place. Our model (see Fig. 3) has two components (C1, C2), whereby

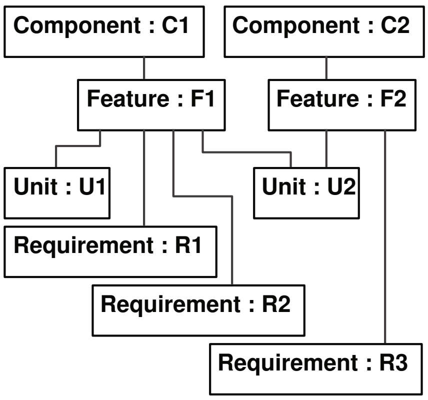

**Fig. 3.** Example Model for risk assessment

each component has one feature (C1 is linked to F1, C2 is linked to F2). Furthermore, we have two Units (U1, U2) and three Requirements (R1, R2, R3). F1 uses U1 and U2 and implements R1 and R2. F2 uses U2 and implements R3.

In the risk assessment phase, first the *ProbabilityCriteria* for the Units U1 and U2 are set. We assume, that the project is in the planning phase and therefore the code complexity is estimated manually, e.g., by software developer or architects because there is no source code available for the automatic determination of code complexity. The functional complexity is determined by a function point analysis. The testability is also estimated manually. In our example, we assume that a low code complexity implies a low value for testability. The following table shows the estimated values for the probability criteria of the units U1 and U2 in our example:

| **Unit** | **ProbabilityCriterion** | $\mathbf{p_j}$ |
|---|---|---|
| U1 | Code Complexity | 8 |
| U1 | Functional Complexity | 9 |
| U1 | Testability | 7 |
| U2 | Code Complexity | 3 |
| U2 | Functional Complexity | 2 |
| U2 | Testability | 2 |

In the next step, the *ImpactCriteria* of the three requirements R1, R2, and R3 are determined manually, e.g., by the responsible product manager or the customer based on the requirements documents. The following table shows the estimated values for the impact criteria of the requirements R1, R2, and R3 in our example:

| **Requirement** | **ImpactCriterion** | $\mathbf{i_j}$ |
|---|---|---|
| R1 | Importance | 9 |
| R1 | Usage | 9 |
| R2 | Importance | 4 |
| R2 | Usage | 3 |
| R3 | Importance | 8 |
| R3 | Usage | 9 |

Additionally, for the determination of the factors the weights attached to the components are needed for the computation of risk values. The weight values are determined manually for the components C1, C2, and C3, e.g., by the project or test manager, and are as follows in our example:

| **Component** | **Criterion** | $\mathbf{w_j}$ |
|---|---|---|
| C1 | Code Complexity | 0.4 |
| C1 | Functional Complexity | 0.7 |
| C1 | Testability | 0.6 |
| C1 | Importance | 0.8 |
| C1 | Usage | 0.5 |
| C2 | Code Complexity | 0.9 |
| C2 | Functional Complexity | 0.3 |
| C2 | Testability | 0.4 |
| C2 | Importance | 0.7 |
| C2 | Usage | 0.7 |

In the last step, the time criteria of the features F1 and F2 have to be set to determine the time factors. Because the example project is in the planning phase, the *TimeCriteria* are set to the initial maximal value of 1, because the change and test history are initially 1 and do not reduce the probability. Thus, the time criteria for the features F1 and F2 are as follows:

| **Feature** | **TimeCriterion** | $\mathbf{t_j}$ |
|---|---|---|
| F1 | Change History | 1.0 |
| F1 | Test History | 1.0 |
| F2 | Change History | 1.0 |
| F2 | Test History | 1.0 |

After the values for criteria and weights have been estimated, the values for the factors are calculated automatically. For feature F1 which is related to the units U1 and U2 (with the corresponding probability factors P1 and P2), and to the requirements R1 and R2 (with the corresponding impact factors I1 and I2), probability, impact and time factors are as follows:

| **ProbabilityFactor** | **Formula** | **P(a)** |
|---|---|---|
| P1 | $(8\cdot0.4+9\cdot0.7+7\cdot0.6)/(0.4+0.7+0.6)$ | 8.1 |
| P2 | $(3\cdot0.4+2\cdot0.7+2\cdot0.6)/(0.4+0.7+0.6)$ | 2.2 |

| **ImpactFactor** | **Formula** | **I(a)** |
|---|---|---|
| I1 | $(9\cdot0.8+9\cdot0.5)/(0.8+0.5)$ | 9.0 |
| I2 | $(4\cdot0.8+3\cdot0.5)/(0.8+0.5)$ | 3.6 |

| **TimeFactor** | **Formula** | **T(a)** |
|---|---|---|
| T1 | $(1.0+1.0)/2$ | 1.0 |

By analogy, the probability, impact, and time factors for feature F2 are as follows:

| **ProbabilityFactor** | **Formula** | **P(a)** |
|---|---|---|
| P2 | $(3{\cdot}0.9+2{\cdot}0.3+2{\cdot}0.4)/(0.9+0.3+0.4)$ | 2.6 |

| **ImpactFactor** | **Formula** | **I(a)** |
|---|---|---|
| I3 | $(8{\cdot}0.7+9{\cdot}0.7)/(0.7+0.7)$ | 8.5 |

| **TimeFactor** | **Formula** | **T(a)** |
|---|---|---|
| T2 | $(1.0+1.0)/2$ | 1.0 |

Based on the probability, impact, and time factors, the risk coefficients R1 and R2 for the features F1 and F2 can be calculated:

| **Risk** | **Formula** | **R(a)** |
|---|---|---|
| R1 | $((8.1+2.2)/2{\cdot}1.0){\cdot}(9.0+3.6)/2$ | 32.4 $(5.2\times6.3)$ |
| R2 | $(2.6{\cdot}1.0){\cdot}8.5$ | 22.1 $(2.6\times8.5)$ |

The risk coefficients for R1 and R2 are shown in Fig. 4. Even though, the *ImpactFactor* of R1 is not as high as the *ImpactFactor* of R2, it is categorized into a higher risk category than R2 because of the high probability that a failure occurs.

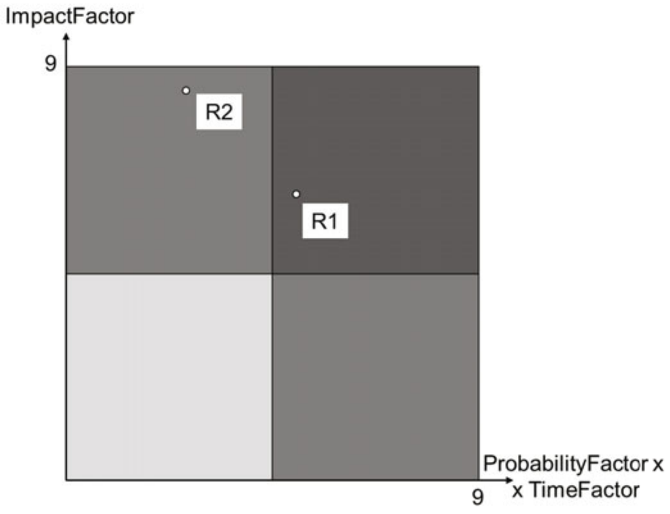

**Fig. 4.** R1 and R2 after first assessment

We assume that the next risk assessment session takes place after the first prototype has been implemented. Furthermore, we suppose, that the different impact and probability criteria have been estimated precisely, such that their values do not change in the actual assessment session. The result of the first risk assessment session was that the risk value of F1 is roughly 10 units higher than F2's risk. Therefore, in the first evaluation phase F1 has been tested more deeply than F2. When testing the features after their implementation, we noticed that F2 has more failures than F1. In our example, we assume that the the value of *failed test cases/number of test cases* for F1 is only 40% of F2's value. Additionally, we assume that it is observed that F2 has been changed more frequently than F1. In the example, F1 has only 60% of the number of updates of F2. Thus, the change history and the test history value for F1 are reduced, but are constant for F2:

| **Feature** | **TimeCriterion** | $\mathbf{t_j}$ |
|---|---|---|
| F1 | Change History | 0.6 |
| F1 | Test History | 0.4 |
| F2 | Change History | 1.0 |
| F2 | Test History | 1.0 |

The recalculated time factors T1 and T2 are then as follows:

| **TimeFactor** | **Formula** | $\mathbf{T(a)}$ |
|---|---|---|
| T1 | (0.6+0.4)/2 | 0.5 |
| T2 | (1.0+1.0)/2 | 1.0 |

As mentioned before, we assume that none of the probability and impact criteria have changed its value, such that the resulting risks coefficients R1 and R2 are as follows:

| **Risk** | **Formula** | $\mathbf{R(a)}$ |
|---|---|---|
| R1 | $((8.1+2.2)/2\cdot0.5)\cdot(9.0+3.6)/2$ | 16.2 $(2.6\times6.3)$ |
| R2 | $(2.6\cdot1.0)\cdot8.5$ | 22.1 $(2.6\times8.5)$ |

The updated risk coefficients and their classification are shown in Fig. 5 which illustrates the effect of the *TimeFactor*, because it can seen, that the *ImpactFactor* of R1 and R2 has not changed but the *ProbabilityFactor* of R1 decreased. Therefore, R1 and R2 are now both in the same risk category and the risk coefficient R1 is lower than the risk coefficient of R2.

With the procedure at hand, resources can be saved by prioritizing the test cases and not implementing and/or executing all of them. Nevertheless, there is an overhead of time and money needed for evaluating the different risk factors. Therefore, if possible most of the criteria should be evaluated automatically. We have categorized criteria according to the degree of automation into *automated*

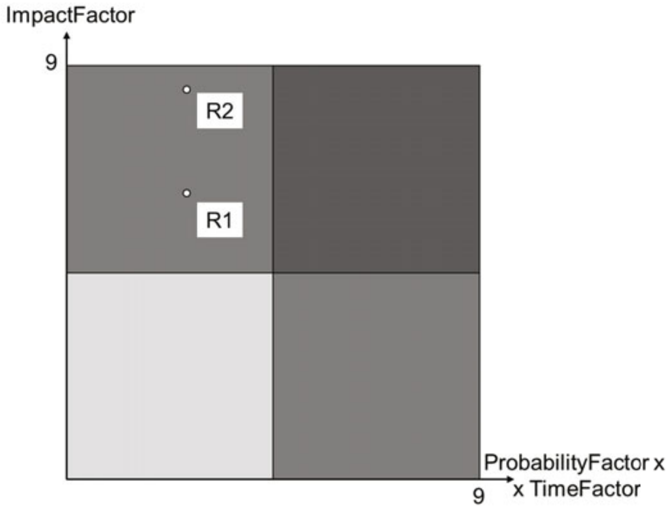

**Fig. 5.** R1 and R2 after second iteration

*evaluation*, *semi-automated evaluation*, and *manual evaluation*. A criterion is evaluated automatically, if all underlying metrics are evaluated automatically, semi-automatically, if all underlying metrics are evaluated automatically or semi-automatically, and manually, if at least one underlying metrics is evaluated manually. Typically code complexity metrics are evaluated automatically, functional complexity metrics are evaluated semi-automatically and customer-related metrics like importance of a product are evaluated manually.

We assume that automatic evaluation is easier to perform and more meaningful for *ProbabilityMetric*s than for *ImpactMetric*s This assumption normally holds because the *ProbabilityMetric*s are derived from formal artifacts like the source code and its architecture for which many analysis tools are available. The *ImpactMetric*s are derived from requirements which are typically only available as informal text and evaluated manually.

The degree of automation of the evaluation can change for specific metrics. For instance, time metrics like the change or test history can only be estimated manually when a project starts but can later be computed automatically based on data from version and test management systems.

Besides the automation, the precision of the assessment of manually and semi-automatically determined metrics can be improved by distribution among the stakeholders. Software developers and software architects may evaluate probability metrics, the project manager sets the component weights, and the impact metrics are evaluated by customers. Time metrics are determined by the test manager before the release and additionally by the customer support after the release of the system.

In the following subsections, we give examples for the automated, semi-automated and manual evaluation of metrics.

### 5.1 Automated Metrics

As automated metrics we consider e.g., *code complexity metrics* because there exist many code metrics, e.g., the number of code lines or the McCabe complexity, that can be determined automatically. In general, we suggest to project the automatically evaluated values to a scale. The lowest value derived, will be projected on the lowest value of the scale, and the highest value on the highest scale value. The other derived values have to be interpolated on the remaining values of the scale. There are many tools for automatically measuring the code complexity, e.g., CAST [30], Understand [31], Sonar [32] or iPlasma [33].

The results of the automatic evaluation can be quantitative or qualitative. Quantitative results are numeric values (see [34] for an overview of typical metrics) like lines of code (LoC), McCabe complexity [22] or depth of the inheritance tree. Qualitative results are verbal description, e.g., of error-prone statements. In this case, the different warnings and error messages have to be quantified. Both types of metrics are derived by automatic static code analysis [35]. Thus, if the evaluation procedure is based on several code metrics, only an algorithm for automatically determining the value of a criterion has to be implemented.

### 5.2 Semi-automated Metrics

Semi-automated metrics are derived automatically but the resulting value is only an approximation, e.g., because relevant information is not taken into account by the derivation algorithm. Therefore, the value has to be corrected manually and the automatically derived value only provides decision support for the user. As an example for semi-automated metrics we consider *functional complexity metrics*. There are several fuzzy metrics for measuring functional complexity, e.g., the length of the textual description of an algorithm, the occurrence or absence of keywords, the complexity of the computation graph of the algorithm, and the computational complexity of the algorithm.

In many cases, none of the mentioned metrics is solely a precise measure for the functional complexity. But they provide valuable input for a subsequent manual estimation of the functional complexity.

### 5.3 Manual Metrics

As a manually determined metrics, e.g., the predicted *usage*, i.e., the frequency of use and importance to user, can be considered. The usage can only be estimated depending on the expected user number of the *System* and the privileges needed to use the specific *Requirement*. Even though the text might contain hints for a possible usage, the real usage only can be estimated with experience and background knowledge. As support for the human estimators, a scale with textual values can be provided. The following table shows such a scale with textual values. The scale defines five values (seldom, sometimes, average, often, highest) which are projected on the numeric values of the metrics.

| **Usage Metrics Ordinal Scale Values** | |
|---|---|
| seldom | 1 |
| sometimes | 3 |
| average | 5 |
| often | 7 |
| highest | 9 |

Another possible interpretation of the usage criterion is shown in the next table, which represents a cumulative scaling function. The selected entries are summed up to determine the final value of the metric.

| **Usage Metrics Cumulative Scale Values** | |
|---|---|
| many different users | +3 |
| barrier-free | +1 |
| often used | +3 |
| untrained staff | +2 |

The different values are defined during the risk identification phase and stakeholders who conduct the risk assessment only have to select the proper values. For the cumulative scale, the sum of all values has to be lower or equal to the maximal scale value (in our case 9). As mentioned before the evaluation type of metrics can change during the application life cycle. For instance, the usage has to be manually estimated before the system is in use, but can be determined automatically in the maintenance phase.

## 5.4 Metric Estimation Workshop

Many metrics have to be evaluated manually or semi-automatically. For the evaluation process, estimation workshops may be conducted. For such workshops where human experts estimate values, it is very helpful if the scales for manually or semi-automatically evaluated metrics do not only consist of numeric values, but additional provide textual descriptions for each value of an ordinal or cumulative scale (see Section 5.3 for examples).

As mentioned before, time metrics are typically computed in an automatic way as soon as version and test management systems are available and are therefore not separately considered in an estimation workshop. In our process the semi-automatically and automatically evaluated probability and impact values are estimated by two separate teams. The impact metrics are estimated by customer-related stakeholders such as product managers or customers themselves, and the probability metrics by technical stakeholders such as software developers or architects. Each member of the particular estimation team gets an estimation form for each requirement or unit for determining probability or impact values. Afterwards, the estimated values of the single team members are compared and discussed by the whole team, until a consensus is achieved and the final value can noted by the moderator of the estimation workshop. It may be useful to document the activities in the estimation workshop because a complicated decision-finding can be an indicator for a high risk. The separated estimation of probability and impact values guarantees an independent and efficient estimation process.

## 6 Conclusion

In this paper we have defined a risk assessment model and a risk assessment procedure based on a generic risk-based testing process. The risk-based testing process consists of the phases risk identification, test planning, test design, evaluation, and risk assessment which is the core activity of the process.

The package risk of the risk assessment model defines factors, criteria, metrics and functions for the computation of risks. Additionally, the model considers requirements, implementation, plus system elements as basis for the risk assessment, and tests to which the resulting risks are attached. The static view of the risk assessment model is the foundation for the dynamic view of the risk assessment procedure that is presented by an example considering impact, probability and time criteria.

The concrete algorithms for the determination of risks are based on manually, semi-automatically, and automatically evaluated metrics. For each type of metrics evaluation we present examples and determination strategies. The integration of manual, semi-automatically and automatic metrics as proposed in our approach significantly improves risk assessment because it supports more efficient determination of metrics by automation and distribution among stakeholders. Efficient risk assessment is the prerequisite for the successful application of risk-based testing in practice.

As future work we evaluate the effort of the risk assessment process empirically by its application to industrial projects. Based on the results of the risk assessment proposed in this paper, we investigate the other activities of the risk-based testing process. The risk classification influences the definition of test methods and test exit criteria in the test strategy. The prioritization of tests, where areas with high risks have higher priority and are tested earlier, is considered for the test design. In the evaluation phase residual risk of software delivery are estimated. All these tasks are evaluated empirically by the application to industrial projects.

**Acknowledgment.** This work is sponsored by the project QE LaB - Living Models for Open Systems (FFG 822740).

## References

1. Merriam-Webster: Merriam-Webster Online Dictionary (2009), `http://www.merriam-webster.com/dictionary/clear` (accessed: July 12, 2011)
2. Bach, J.: Troubleshooting risk-based testing. Software Testing and Quality Engineering 5(3), 28–33 (2003)

3. Ropponen, J., Lyytinen, K.: Components of software development risk: How to address them? a project manager survey. IEEE Transactions on Software Engineering 26(2), 98–112 (2000)
4. Pfleeger, S.: Risky business: what we have yet to learn about risk management. Journal of Systems and Software 53(3), 265–273 (2000)
5. Boehm, B.: A spiral model of software development and enhancement. Computer 21(5), 61–72 (1988)
6. Kontio, J.: Risk management in software development: a technology overview and the riskit method. In: Proceedings of the 21st International Conference on Software Engineering, pp. 679–680. ACM (1999)
7. Karolak, D., Karolak, N.: Software Engineering Risk Management: A Just-in-Time Approach. IEEE Computer Society Press, Los Alamitos (1995)
8. Amland, S.: Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. Journal of Systems and Software 53(3), 287–295 (2000)
9. Bach, J.: Heuristic risk-based testing. Software Testing and Quality Engineering Magazine 11, 99 (1999)
10. Carr, M., Konda, S., Monarch, I., Ulrich, F., Walker, C.: Taxonomy-based risk identification. Carnegie-Mellon University of Pittsburgh (1993)
11. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of software Test. ACM (2008)
12. Stallbaum, H., Metzger, A.: Employing Requirements Metrics for Automating Early Risk Assessment. In: Proc. of MeReP 2007, Palma de Mallorca, Spain, pp. 1–12 (2007)
13. Lund, M.S., Solhaug, B., Stolen, K.: Model-driven Risk Analysis. Springer, Heidelberg (2011)
14. Lee, W., Grosh, D., Tillman, F.: Fault tree analysis, methods, and applications - a review. IEEE Transactions on Reliability (1985)
15. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006)
16. Alexander, I.: Misuse cases: Use cases with hostile intent. IEEE Software 20(1), 58–66 (2003)
17. Asnar, Y., Giorgini, P.: Modelling Risk and Identifying Countermeasure in Organizations. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 55–66. Springer, Heidelberg (2006)
18. McCall, J., Richards, P.K., Walters, G.F.: Factors in software quality. Technical report, NTIS, Vol 1, 2 and 3 (1997)
19. Haimes, Y.Y.: Risk Modeling, Assessment, and Management. Wiley (2004)
20. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th International Conference on Software Engineering. ACM (2006)
21. Illes-Seifert, T., Paech, B.: Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs. Information and Software Technology 52(5) (2010)
22. McCabe, T.: A complexity measure. IEEE Transactions on software Engineering, 308–320 (1976)
23. Jiang, Y., Cuki, B., Menzies, T., Bartlow, N.: Comparing design and code metrics for software quality prediction. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, pp. 11–18. ACM (2008)

24. NIST: National Vulnerability Database, http://nvd.nist.gov/ (accessed: July 12, 2011)
25. The Open Source Vulnerability Database, http://osvdb.org/ (accessed: July 12, 2011)
26. Frei, S., May, M., Fiedler, U., Plattner, B.: Large-scale vulnerability analysis. In: Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense, pp. 131–138. ACM (2006)
27. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. IEEE Security & Privacy 4(6), 85–89 (2006)
28. Spillner, A., Linz, T., Rossner, T., Winter, M.: Software Testing Practice: Test Management. Dpunkt (2007)
29. van Veenendaal, E.: Practical risk–based testing, product risk management: the prisma method. Technical report, Improve Quality Services BV (2009)
30. CAST, http://www.castsoftware.com/ (accessed: July 12, 2011)
31. Understand, http://www.scitools.com/ (accessed: July 12, 2011)
32. Sonar, http://www.sonarsource.org/ (accessed: July 12, 2011)
33. iPlasma, http://loose.upt.ro/iplasma/index.html (accessed: July 12, 2011)
34. Zhao, M., Ohlsson, N., Wohlin, C., Xie, M.: A comparison between software design and code metrics for the prediction of software fault content. Information and Software Technology 40(14), 801–810 (1998)
35. Nagappan, N., Ball, T.: Static analysis tools as early indicators of pre-release defect density. In: Proceedings of the 27th International Conference on Software Engineering, pp. 580–586. ACM (2005)

# Inspection and Test Process Integration Based on Explicit Test Prioritization Strategies

Frank Elberzhager[1], Alla Rosbach[1], Jürgen Münch[2], and Robert Eschbach[1]

[1] Fraunhofer IESE, Fraunhofer Platz 1,
67663 Kaiserslautern, Germany
{frank.elberzhager,alla.rosbach,
robert.eschbach}@iese.fraunhofer.de
[2] University of Helsinki, P.O. Box 68,
00014 Helsinki, Finland
juergen.muench@cs.helsinki.fi

**Abstract.** Today's software quality assurance techniques are often applied in isolation. Consequently, synergies resulting from systematically integrating different quality assurance activities are often not exploited. Such combinations promise benefits, such as a reduction in quality assurance effort or higher defect detection rates. The integration of inspection and testing, for instance, can be used to guide testing activities. For example, testing activities can be focused on defect-prone parts based upon inspection results. Existing approaches for predicting defect-prone parts do not make systematic use of the results from inspections. This article gives an overview of an integrated inspection and testing approach, and presents a preliminary case study aiming at verifying a study design for evaluating the approach. First results from this preliminary case study indicate that synergies resulting from the integration of inspection and testing might exist, and show a trend that testing activities could be guided based on inspection results.

**Keywords:** software inspections, testing, quality assurance, integration, focusing, synergy effects, case study, study design.

## 1 Introduction

Quality assurance activities, such as inspection (i.e., static quality assurance) and testing (i.e., dynamic quality assurance) activities, are an essential part of today's software development in order to ensure software products of high quality. However, the costs for performing quality assurance activities can consume more than 50 percent of the overall development effort, especially for testing [7]. Moreover, it is often unclear how to systematically guide and focus testing activities.

Existing approaches to focusing testing activities are widely based on metrics such as size or complexity, gathered from the development of current or historical software products. Regarding inspection and testing activities, systematic integration is often missing. Inspection and testing activities are usually applied in sequence, i.e., in isolation, and do not exploit synergy effects such as reduced effort or the use of inspection results to guide testing activities.

This article presents an integrated inspection and testing approach that is able to guide testing activities based on inspection results. Parts of a system that are expected to be most defect-prone or defect types that are especially relevant can be prioritized based on defect data gathered during an inspection. In order to be able to conduct a focused testing activity, knowledge about relationships between inspections and testing is required. Such relationships are usually context-specific. Therefore, it is necessary to prove whether reliable evidence about such relationships exists in a given context. If no such evidence exists, assumptions need to be made regarding the relationships between inspection and testing activities. For example, one assumption might be that parts of a system where a significant number of defects are found by an inspection contain more defects to be found by testing (i.e., a Pareto distribution of defects is expected). Such assumptions have to be evaluated in a given context in order to provide appropriate guidance for testing.

A study design was determined and verified during a preliminary case study in which the integrated inspection and testing approach was applied. The results showed, for instance, that inspection and testing activities should focus on defect types that are most suitable for detection (e.g., maintainability problems during inspections, and usability problems during testing), and that an effort reduction for testing of up to 23% was achievable in the given context. However, one important prerequisite for the applicability of the approach is the testability of the software under test.

The remainder of this article is structured as follows: Section 2 presents a short overview of the integrated inspection and testing approach. The study design and the preliminary case study are described in Section 3. Finally, Section 4 concludes the article and gives an outlook on future work. An extended version of this article includes related work and a more detailed description of the approach [5].

## 2 Approach

The main idea of the integrated inspection and testing approach [2], [3] is to use inspection defect data to guide testing activities. In doing so, parts of a system under test that are expected to be most defect-prone or defect types that are expected to show up during testing can be prioritized based on an inspection defect profile (consisting of, for example, quantitative defect data and defect type information from an inspection). The approach is able to prioritize parts of a system or defect types (1-stage approach), or both (2-stage approach), and can thus define a test strategy.

In order to be able to focus testing activities, it is necessary to describe relationships between defects found in the inspection and the remaining defect distribution in the system under test, which also counts for defect types. Consequently, assumptions are explicitly defined. One example of an assumption is that for parts of a system where many inspection defects are found, more defects are expected to be found with testing activities (i.e., a Pareto distribution of defects is expected). Assumptions should be at least grounded on explicitly described hypotheses to make them reliable. Nevertheless, each assumption has to be validated in a given environment in order to be able to decide whether the assumption can be accepted or not (and thus, checking whether valuable guidance for testing activities is provided).

In addition, context factors have to be considered, such as the number of available inspectors or the experience of the inspectors. For example, consider the number of available inspectors and time as two context factors. If only one inspector is available for inspecting certain parts of a system within a limited amount of time, fewer parts can be inspected. Consequently, more effort should be expended on testing activities.

Since an assumption is often too coarse-grained to be applied directly, concrete selection rules have to be derived in order to be operational. For example, the assumption regarding the Pareto distribution of defects can be refined in terms of application level and thresholds, leading to the following exemplary selection rule: "Focus a unit testing activity on code modules where the inspection found more than 10 major defects per 1,000 lines of code".

In addition to the inspection defect profile, metrics and historical data can be combined with inspection defect data in order to improve the prediction of defect-proneness and relevant defect types, and thus, to obtain improved guidance for testing activities. Fig. 1 presents the concrete process steps for guiding testing activities based on inspection results.

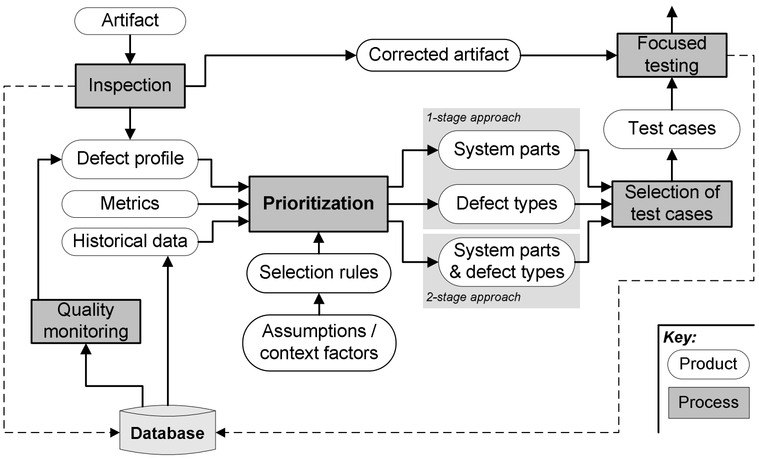

**Fig. 1.** Integrated inspection and testing approach

## 3 Case Study

### 3.1 Goals

The primary goals of the preliminary study were to check the design of the study and to evaluate whether an integrated inspection and testing approach is able to guide testing activities based on inspection results. Inspection defect data should be used to predict those parts of a system under test that remain especially defect-prone and should therefore be addressed by additional testing activities. In addition, defect types should be prioritized for testing based on inspection data.

While the initial case study showed first insights regarding the relationship between inspection and testing activities on the code level, more data should be gained in the preliminary case study presented here. Therefore, different assumptions in a given context describing the relationship between defects found during inspection and testing had to be evaluated regarding their suitability for guiding (i.e., focusing) testing activities.

The following research questions are derived from the primary goals:

- RQ1: Is an evaluation of the integrated approach possible with the given study design, and which assumptions between inspection and testing activities are most suitable in the given context for guiding testing activities?
    - RQ1.1: How appropriate is it to focus testing activities on certain parts of a product based on inspection defect results?
    - RQ1.2: How appropriate is it to focus testing activities on specific defect types based on inspection defect results?

## 3.2 Main Results from Another Evaluation

The preliminary case study presented in this article is similar to an earlier case study [2], [3]. It could be shown that assumptions regarding a Pareto distribution of defects led to suitable predictions of defect-prone parts, while combining inspection results and product metrics led to inconsistent results for the prediction of defect-prone code classes. The main differences of the preliminary study presented here compared to the previous one are that new assumptions are defined, that the integrated approach is evaluated in a different context (e.g., another product that was inspected and tested, new subjects), and that system testing is considered. Furthermore, the preliminary study presented here has a special focus on evaluating the design of the study before applying the approach in an industrial environment.

## 3.3 Context

A Java tool called DETECT (dependability focused inspection tool) was used for evaluating the integrated inspection and testing approach. The tool supports people who perform an inspection. Currently, it mainly supports individual defect detection with the help of different kinds of reading support and allows defining new checklists for use during defect detection. The different kinds of reading support include different tree structures and two kinds of checklists. The tool provides a three-part view for the inspector: a tracking mode that documents each step; the artifact to be checked; and the corresponding reading support (e.g., a checklist).

The tool was mainly developed by one developer. Currently, it consists of about 57k lines of code (without blank lines and comments), about 380 classes, and about 2,300 methods. The developer identified the critical code parts that should be inspected and discussed the selection of the code classes with the inspection team. In order to be able to finish the inspection within existing time constraints, it was decided to inspect only one kind of reading support, namely GITs (goal-indicator trees [6]). Overall, six inspectors checked 12 code classes, comprising about 7,300 lines of code. Each inspector checked four code classes, consisting of about 2,500 lines of code.

Table 1 shows the experience, respectively the knowledge, of the six inspectors regarding the inspection, the reading support to be checked, and the code structures (i.e., programming knowledge). Three values (low, middle, high) are used for the classification. Finally, the assigned checklists are shown.

The testing activities were performed by the developer of the tool and one additional tester. Neither one was involved in the inspection.

**Table 1.** Experience of inspectors and assigned checklists

| No. | Inspection knowledge | GIT knowledge | Programming knowledge | Defect detection focus |
|---|---|---|---|---|
| 1 | + | ++ | ++ | requirements |
| 2 | ++ | ++ | ++ | requirements |
| 3 | + | o | ++ | implementation |
| 4 | ++ | + | ++ | implementation, reliability |
| 5 | ++ | o | o | code documentation |
| 6 | ++ | ++ | + | code documentation |

## 3.4 Design

The preliminary study described in this article followed a similar design compared to the first evaluation of the integrated inspection and testing approach [2], [3].

First, a code inspection was conducted by six computer scientists (step 1) using checklists. Overall, four different checklists were used, addressing requirements fulfillment, implementation, reliability, and code documentation. Each checklist consisted of three to eight questions and was assigned to those inspectors who could answer the questions effectively. Finally, the checklists were mapped to the relevant code classes by the developer of the tool so that each inspector checked four code classes. One experienced quality assurance engineer aggregated the findings from all inspectors. The developer analyzed each problem and decided whether a real defect was found that had to be corrected or whether problems that were documented by an inspector were only due to a misunderstanding and could be removed without any correction.

The next step was the quality monitoring of the resulting inspection defect profile (step 2). Reading rate, overall number of found defects, and defect distribution were considered.

Step 3 comprised the prioritization, i.e., a prediction of defect-prone parts and defect types had to be conducted. For this, four context-specific assumptions were determined that were to be evaluated. Two assumptions of the initial study [2], [3] were reused, and two new assumptions were defined based on experiences made during the first study.

Finally, selecting test cases and conducting focused testing activities would be the last steps (step 4 and 5). However, in order to be able to evaluate the stated assumptions, prioritized as well as not prioritized parts were tested by two testers.

This enabled a detailed analysis of the assumptions regarding their appropriateness. First, a unit test of code classes was started. Test cases were derived using equivalence partitioning. Code classes that had been inspected and some additional ones identified as being most critical or important were selected for testing. However, it turned out that efficient unit testing was not possible due to bad testability of the code classes. The code structure did not suit the unit test approach (e.g., due to anonymous inner classes, anonymous threads, private fields and methods). To neutralize the problems of the code structure, mocking frameworks (i.e., a simulation of the behavior of code classes) were used. However, this framework turned out to be very complex for inexperienced testers.

Beside unit testing, a manual system test was conducted in order to analyze whether prioritization is possible between different levels (i.e., using defect information from the code level to guide tests for the system level). System tests were derived through typical walkthrough scenarios that followed the main functionality the tool offers. Afterwards, the results from this testing activity were used as a baseline and compared to the prioritization when the defined assumptions were evaluated.

## 3.5 Execution

**Step 1: Performing the inspection**

Before the inspection was performed, a team meeting was held where the checklists were explained and an overview of the code to be inspected was presented. Afterwards, each inspector checked the assigned code classes with the assigned checklist and documented all findings and the place of occurrence in a problem list. In addition, defect type and defect severity were recorded. Each code class was checked by at least two inspectors. Overall, 1450 minutes were spent on individual defect detection (ranging from 90 to 280 minutes consumed per inspector).

**Table 2.** Defect content and defect density of each inspected code class

| Code class | *I* | *II* | *III* | *IV* | *V* | *VI* | *VII* | *VIII* | *IX* | *X* | *XI* | *XII* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Defect content | 4 | 18 | 19 | 2 | 34 | 18 | 13 | 24 | 31 | 11 | 10 | 5 |
| Defect density | .009 | .021 | .020 | .008 | .061 | .057 | .038 | .031 | .045 | .026 | .031 | .016 |

One experienced quality assurance engineer compiled the defect detection profile and the developer of the tool checked for each defect whether it has to be corrected or not. Of 236 problems found in total, 189 defects to be corrected remained. Table 2 shows the defect content (absolute number of defects) and defect density (absolute number of defects divided by lines of code) of the twelve inspected code classes. Table 3 shows a sorted list of the ODC-classified defects [8]. 54 defects (e.g., unclear or missing comments) could not be classified according to any of the existing defect types.

**Table 3.** ODC-classified defects from inspection

| ODC defect types | Sub-total | ODC defect types | Sub-total | **Total** |
|---|---|---|---|---|
| algorithm / method | 53 | relationship | 1 | |
| checking | 36 | timing / serialization | 0 | |
| function / class / object | 32 | interface / o-o messages | 0 | |
| assignment / initialization | 13 | other | 54 | |
| Sub-total | 134 | | 55 | **189** |

**Step 2: Monitoring the inspection results**

Because this was the first systematic quality assurance run of the DETECT tool, no historical data was available that could be used for monitoring the inspection results. Instead, data from the first study of the integrated inspection and testing approach [2], [3] was used since the environment was similar. In addition, data from the literature was considered. The reading rate was about 630 lines of code per hour, which is similar to the first study (there, it was 550, respectively 685, lines of code per hour in two quality assurance runs). The number is rather high compared to reading rates recommended in the literature, but consistent with experiences from industry [1]. Some reasons for the high number are that all lines of code were counted (including blank lines and comments) and that the individual checklists pinpointed the inspectors to certain parts, whereas other parts were read faster. Finally, the overall number of found defects seemed reasonable compared to the first study and the distribution of minor, major, and crash defects was also similar to the first study.

**Step 3: Prioritizing the testing activities**

In order to guide testing activities, a prediction of defect-prone parts and defects of those defect types that are expected to appear during testing was made, i.e., those parts and defect types were prioritized. Four assumptions were stated, including instructions for the prioritization. More details and explanations can be found in [2], [3], [4].

*Assumption 1:* If no defined selection criterion is used to determine parts of a system that should be inspected, it is expected that a significant number of defects still remain in those parts that are not inspected (i.e., an equal distribution of defects is assumed). Consequently, testing should be focused especially on those uninspected parts of a system.

*Assumption 2:* Parts of a system where a large number of inspection defects are found indicate more defects to be found with testing (i.e., a Pareto distribution of defects is assumed). Consequently, testing should be focused especially on those inspected parts of a system that were particularly defect-prone.

*Assumption 3:* Inspection and testing activities find defects of various defect types with different effectiveness. For inspections, this includes, e.g., maintainability problems. For testing, this includes, e.g., performance problems. Consequently, inspection and testing activities should be focused on those defect types that are most convenient to find.

*Assumption 4:* Defects of the defect types that are found most often by the inspection (i.e., a Pareto distribution of defects of certain defect types is assumed) indicate more defects of the defect types to be found with testing. Thus, testing should be focused on those defect types that the inspection identified most often.

A derivation of concrete selection rules is skipped here. However, this can be done easily using the inspection defect profile; examples of concrete selection rules are shown in [2], [3], and some applied selection rules are shown in Section 3.6.

**Step 4 and 5: Selecting test cases and conducting the testing activities**

To evaluate the integrated inspection and testing approach and the stated assumptions, testing activities were performed without considering the inspection defect profile for the prioritization (however, the inspection defects were corrected before testing activities started). 40, respectively 42, similar test cases were applied during system test by the two testers, covering the main functionality of the tool, i.e., different kinds of reading support, the interaction of reading support and an artifact to be inspected, the generation of a report of the findings, and creating a checklist was tested. In addition, some explorative testing was performed by the tester that did not develop the tool.

**Table 4.** Test results from system testing

| | Number of test cases | | Number of defects found | | Defect ids | | Effort (min) | |
|---|---|---|---|---|---|---|---|---|
| **Tested functionality** | tester 1 | tester 2 | tester 1 | tester 2 | tester 1 | tester 2 | tester 1 | tester 2 |
| reading support: GIT | 3 | 3 | 1 | 1 | id1, id8* | id1 | 10 | 6 |
| reading support: SGIT | 3 | 3 | 0 | 1 | id9* | id1 | 7 | 6 |
| reading support: GC | 3 | 3 | 0 | 0 | id10* | | 7 | 6 |
| reading support: VID | 0 | 11 | 0 | 1 | id11* | id1 | 0 | 30 |
| reading support: CL | 1 | 1 | 0 | 0 | | | 3 | 2 |
| interaction | 15 | 8 | 5 | 2 | id2, id3, id4, id6, id7, id12* | id2, id3 | 33 | 21 |
| report generation | 1 | 1 | 1 | 0 | id5, id13* | | 15 | 10 |
| checklist creation | 16 | 10 | 1 | 0 | id4 | | 40 | 10 |

During the system test, seven additional defects regarding functionality were found by the two testers. Running the defined test cases took about 90, respectively 120 minutes. In addition, effort for explorative testing, test documentation, debugging, and correction was consumed, resulting in an overall test effort for both testers of about 14 hours. The distribution of defects with respect to functionality can be found in Table 4 (id1 – id7). Tester 1 found one defect (defect id 1) when testing the GIT reading support (which was inspected on the code level). However, this defect is independent of the concrete reading support. Tester 2 also found this defect when testing GITs, but also when testing the other tree-based reading model SGITs or VIDs. Furthermore, most of the defects occurred when testing the interaction between reading support and the artifact view. Two more defects were found when testing checklist creation and report generation. In addition, tester 1 found six more usability problems that were equally distributed (id8* – id13*), i.e., for almost each functionality tested, one usability problem was found.

## 3.6 Results of the Case Study and Lessons Learned

**RQ1.1:** Our first objective was to check whether the inspection defect information could be used to predict defect-proneness within code classes in order to focus unit testing activities. Unfortunately, the unit test activity could not be completed due to bad testability of the code and no new defects were found. Therefore, research question 1.1 could not be answered with respect to the unit level. Instead, the system test activity was used to analyze whether the code inspection results can provide valuable predictions for focusing system testing. Assumptions one and two were applied accordingly. We were aware that this prioritization would mean a different level of granularity, because for system tests it is not possible to address certain code classes; rather, they are used to address functionalities.

Five different kinds of reading support and three additional tool functionalities were tested and revealed that most of the defects were found in parts that had not been inspected. One functional defect was found when applying the GIT reading support (which was also inspected); however, this defect occurred independently of the concrete reading support and was also found when testing other kinds of reading support. Therefore, assumption one indicates a trend towards an appropriate prioritization, respectively prediction, of defect-proneness and might help in guiding system testing activities with reduced effort. Considering only the test execution effort, an effort reduction of between 8% and 23% could be achieved, depending on the concrete selection rules used. When defining a selection rule omitting the GIT test cases, this leads to the lowest reduction in the number of test cases, while all functional defects are found. Omitting SGITs as well, which are a very similar form of reading support, increases the saved effort. In addition, omitting test cases for checklists (i.e., low-complexity reading support), an effort reduction of up to 23% is achievable, with all functional defects still being found. However, the absolute numbers for conducting the tests are rather low and test derivation, documentation, and further activities are not considered here. Consequently, the numbers have to be treated with caution.

With respect to the evaluation of the study design, it is essential that appropriate testability exists in order to focus the testing activities on the same system level.

**RQ1.2:** Our second objective was to analyze the relationship between defect types identified in the inspection and during testing. Considering assumption three, many of those inspection defects classified as 'other' were documentation problems (e.g., missing explanation of a method, unclear description). Such kinds of defects affect the maintainability of the product under test and are not detectable with testing, since they do not influence the observable functionality. Regarding testing, six additional usability problems were found by a tester (e.g., bad readability of parts of reading support). Such kinds of problems can be identified if a graphical user interface is used during testing, but are usually not found during the inspection.

In terms of maintainability and usability, it is rather easy to dedicate them to inspection respectively testing activities in order to find such problems. However,

**Table 5.** ODC-classified defects from inspection and system testing

| ODC defect types | Inspection | Testing |
|---|---|---|
| algorithm / method | 53 | 2 |
| checking | 36 | 4 |
| function / class / object | 32 | 0 |
| assignment / initialization | 13 | 0 |
| relationship | 1 | 1 |
| timing / serialization | 0 | 0 |
| interface / o-o messages | 0 | 0 |
| other | 54 | 6 |
| **Total** | **189** | **13** |

with respect to the ODC classification used for the inspection defects, detecting a relationship to defects found during the system test is difficult due to the difference in granularity between code defect types and system defect types. A post-mortem analysis of the seven functional defects found during testing revealed that most of them were classified as checking or algorithm / method defects, which fits exactly with the two defect types identified most often during inspections (see Table 5). Nevertheless, it is still unclear whether it is possible to derive system tests in a systematic manner that can address such kinds of problems and how this could be done. It might be possible that a defect classification, such as the ODC, is not suitable for guiding system test activities. An explorative study for identifying an appropriate defect classification would be necessary in this case. Finally, due to an uncompleted unit testing activity, no new insights regarding relationships between inspection defect types and testing defect types could be obtained on the unit level.

**RQ1:** To conclude the main results for research question one, a trend was found that testing activities might be guided based on inspection results, i.e., defect-prone parts and defect types could be predicted based on inspection defect data, and testing activities could be focused on certain parts and defect types. However, the quality of such focusing depends on the assumptions made in the given context. In our context, parts that had not been inspected contained additional defects that were found during testing. However, this can only be stated for defects found during system testing because unit testing could not be fully completed (which shows the importance of good testability). Therefore, appropriate testability is an inevitable prerequisite to performing a suitable evaluation, respectively application, of the integrated approach. An effort reduction for test case execution of up to 23% could be achieved when focusing on parts of the system, with the same level of quality being achieved. With respect to defect types, especially maintainability defects were found during inspections, while usability problems were found during testing.

The initial results of the preliminary study presented in this article together with results from the initial case study indicated a first trend towards the applicability of the approach and the potential for exploiting further synergy effects when integrating inspection and testing processes. As inspection and testing processes are established quality assurance activities that are widely applied in industry, and effort reduction is both a current and a future challenge, using inspection results to focus testing activities is a promising approach to be considered in order to address this challenge.

### 3.7 Threats to Validity

Next, we discuss what we consider to be the most relevant threats to validity.

**Conclusion Validity:** The number of testers and the number of found test defects was low. One reason might be the low experience regarding testing. Consequently, no statistically significant data could be obtained.

**Construct Validity:** To demonstrate the integrated approach, different assumptions were derived in our context. Four assumptions were used and analyzed regarding their suitability. However, more assumptions are reasonable and may lead to better or worse predictions. Finally, the selection of ODC was reasonable when focusing on the unit test level, but may not be suitable when using it for the system test.

**Internal Validity:** The subjects selected for the inspection and for the testing activity may have influenced the number of defects that were found. However, the effect was slightly reduced by using checklists that focus an inspector on certain aspects in the code and by using equivalence partitioning, respectively addressing the main functionality, for the testing activity. Finally, defects could be classified differently.

**External Validity:** The DETECT tool inspected and tested in the preliminary study is one example to which the integrated inspection and testing approach was applied. Few test defects were found that could be used for the analysis of the assumptions. Larger software, as typically developed by software companies, is expected to result in more test defects to be found. Assumptions have to be evaluated anew in each new environment, meaning that the conclusions drawn with respect to the used assumptions cannot be generalized directly. Finally, the results can only be transferred to a context where a comparable number of defects are found.

## 4 Summary and Outlook

To address the challenge of guiding testing activities, an integrated inspection and testing approach was presented that is used to predict defect-prone parts of a system and defect types of relevance in order to focus testing activities based on inspection results. This requires explicitly defined assumptions describing the relationships between inspection defects and testing defects. A preliminary case study was presented that analyzed four different assumptions in a given context. First trends could be seen regarding a prediction of defect-proneness. In addition, inspection and testing activities should be focused on those defect types that are most convenient to find, e.g., addressing maintainability problems during inspections, and usability problems during testing. It is worth noting that assumptions are probably not generally acceptable and thus, have to be re-evaluated in each new context in order to obtain evidence on them.

From an industry point of view, the integrated inspection and testing approach can lead to several benefits. Improvements in effectiveness and efficiency are often goals with respect to quality assurance activities. The approach might be applied in order to reduce test effort or to find more defects by focusing the available test effort on parts that are expected to be most defect-prone based on the inspection results. Furthermore, detailed knowledge about inspection and test relationships can lead to an improved overall quality assurance strategy and support the balancing of inspection and testing activities.

With respect to future work, an improvement of the approach and additional evaluations are planned. Based on the preliminary study and the evaluation of the study design, we got new insights into what is necessary with respect to a sound evaluation of the approach (e.g., appropriate testability). We recently began an analysis of inspection and test defect data from an industrial context in order to prove several of our assumptions as well as the potential for effort savings. The knowledge from this study will be incorporated.

Regarding the improvement of the integrated inspection and testing approach, more guidance on how to derive assumptions in a systematic manner should be defined. Furthermore, results from different inspection activities or inspections of only parts of an artifact might be used for guiding testing activities.

**Acknowledgments.** This work has been funded by the Stiftung Rheinland-Pfalz für Innovation project “Qualitäts-KIT” (grant: 925). We would also like to thank Stephan Kremer for tool development, all participants of the study, and Sonnhild Namingha for proofreading.

## References

[1] Cohen, J.: Best kept Secrets of Peer Code Review: Code Reviews at Cisco Systems, 63–87 (2006)
[2] Elberzhager, F., Eschbach, R., Muench, J.: Using Inspection Results for Prioritizing Test Activities. In: 21st International Symposium on Software Reliability Engineering, Supplemental Proceedings, pp. 263–272 (2010), http://inspection.iese.de
[3] Elberzhager, F., Muench, J., Rombach, D., Freimut, F.: Optimizing Cost and Quality by Integrating Inspection and Test Processes. In: International Conference on Software and Systems Process, pp. 3–12 (2011), http://inspection.iese.de
[4] Elberzhager, F., Eschbach, R., Muench, J.: The Relevance of Assumptions and Context Factors for the Integration of Inspections and Testing. In: 37th Euromicro Software Engineering and Advanced Application, Software Product and Process Improvement, pp. 388–391 (2011)
[5] Elberzhager, F., Eschbach, R., Rosbach, A., Münch, J.: Inspection and Test Process Integration based on Explicit Test Prioritization Strategies, IESE Report (2011)
[6] Elberzhager, F., Eschbach, R., Kloos, J.: Indicator-based Inspections: A Risk-oriented Quality Assurance Approach for Dependable Systems. In: Software Engineering 2010, GI edn. Lecture Notes in Informatics, vol. 159, pp. 105–116 (2010)
[7] Harrold, M.J.: Testing: A Roadmap. In: International Conference on Software Engineering. The Future of Software Engineering, pp. 61–72 (2000)
[8] Orthogonal Defect Classification, IBM, http://www.research.ibm.com/softeng/ODC/ODC.HTM

# Towards a Security and Dependability Pattern Development Technique for Resource Constrained Embedded Systems

Nicolas Desnos, Brahim Hamid, Christian Percebois, and Damien Gouteux

IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9 France
{desnos,hamid,percebois,gouteux}@irit.fr

**Abstract.** Security and Dependability (S&D) becomes a strong requirement even in resource constraint embedded systems (RCES). Many domains are not traditionally involved in this kind of issue and have to adapt theirs current processes.

RCES development with S&D requirements represents a complex task. On one hand, it is necessary to provide a solution which manages the cohabitation of different concerns (hardware, software, security specialists, etc). On the other hand, it is advantageous to assist S&D specialists by providing building blocks corresponding to their needs. Of course, guidelines will help them during all the application development life cycle.

A solution based on model driven engineering (MDE) and patterns seems promising in order to resolve this issue. MDE authorizes different businesses to work together with a higher usual abstraction level. Then, patterns can provide practical solutions to meet specific requirements. This paper presents a new pattern development techniques to be inserted in a MDE-based solution. The approach uses a model as its first structural citizen along the course of the development process of S&D patterns for trusted RCES applications.

**Keywords:** Resource Constrained Embedded Systems, Security, Dependability, Pattern, Meta-model, Model Driven Engineering.

## 1 Introduction

An embedded system [24,13,15] is a system that is composed of two main parts, software and hardware, which evolves in a real world environment. An embedded system is an information processing systems integrating *HW* and *SW* embedded into enclosing products to fulfill a specific function. Such systems come with a large number of common characteristics, including real-time and temperature constraints, dependability as well as efficiency requirements. Embedded systems are not classical software, which can be built with usual paradigms.

Consequently, their non-functional requirements such as security and dependability (S&D) [18] become more important as well as more difficult to achieve. Embedded systems solutions are expected to be efficient, flexible, reusable on rapidly evolving hardware and of course at low cost [19]. Solutions are usually concrete technologies for a specific domain (avionics, automotive, transports and energy).

Model-Driven Engineering (MDE) provides a very useful contribution for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. It helps the designer to specify in a separate way non-functional requirements such as security and/or dependability issues at an even greater level that are very important to guide the implementation process. Of course, a MDE approach is not sufficient but offers an ideal development context. While using a MDE framework, it is possible to help S&D specialists in their task. Indeed, it would be interesting to suggest solutions and to guide them according to their S&D requirements. In software engineering, patterns meet this need. The goal of the paper is to propose a new pattern development technique in order to make easy their use in a building process of software applications with S&D support.

The concept of pattern was first introduced by Alexander [3]. A pattern deals with a specific, recurring problem in the design or implementation of a software system. It captures expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, components implementations, or frameworks cannot. With regard to S&D aspects, Yoder and Barcalow [22] were the first to work on security pattern documentation. Unfortunately, most of S&D patterns are expressed in a textual form, as informal indications on how to solve some security problems. Some of them use more precise representations based on UML diagrams, but these patterns do not include sufficient semantic descriptions in order to automate their processing and to extend their use. Furthermore, there is no guarantee of the correct application of a pattern because the description does not consider the effects of interactions, adaptation and combination.

The main purpose of this paper is to propose model-based S&D patterns to get a common representation of patterns for several domains in the context of resource constrained embedded systems (RCES). This domain refers to systems which have memory, computational processing power constraints and/or limited energy constraints. The solution considered here is based on meta-modeling techniques to encode S&D patterns at even greater level of abstraction. Hence, we will present an approach using a model as its first structural citizen all along the development process of S&D patterns for trusted RCES applications. As a result, S&D patterns will be used as brick to build trusted applications through a model driven engineering approach.

The rest of this paper is organized as follows. The next section presents the approach. Based on three level of abstraction, a pattern development technique with S&D properties is detailed. In Section 3 we examine a test case that has several S&D requirements: Secure Service Discovery. In Section 4, we review most related work that address pattern development and S&D patterns. Finally, Section 5 concludes this paper with a short discussion about future works.

## 2 S&D Pattern Development Life Cycle

The main difficulty to overcome S&D patterns in RCES is how to avoid the cost of building a pattern for each S&D properties and/or for each domain. Classical pattern representation is often based on a textual description and pattern structure is rigid (not possible to capture S&D properties). For that, we need a flexible pattern which can be specialized.

The key ideas of the approach presented here are: To capture appropriate characteristics of security and dependability patterns in RCES and to utilize several views. By applying these ideas, we obtain a new engineering process to develop S&D patterns with the desired complexity of use. The technique, as shown in Fig. 1, is based on three levels of abstraction: At the pattern fundamental structure level, a meta-model defines a flexible structure of patterns with S&D and RCES properties. Then, it is possible to specify and develop patterns by a model. First, the pattern model is domain independent and, finally, it is refined for adding domain specific informations. Each level is discussed in the next sub-sections. We sketch a few notable example next. For clarity's sake, we use UML notations to describe such levels.

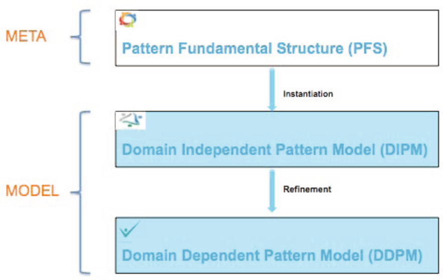
META
Pattern Fundamental Structure (PFS)
Instantiation
Domain Independent Pattern Model (DIPM)
MODEL
Refinement
Domain Dependent Pattern Model (DDPM)

**Fig. 1.** Pattern Development Life Cycle

## 2.1 Pattern Fundamental Structure – PFS

The Pattern Fundamental Structure (PFS) is a meta-model which defines a new formalism for defining S&D patterns. The PFS is presented in the left side of Fig. 2. The originality of this approach is to consider patterns as building blocks that expose services (via interfaces) and manage S&D and RCES properties (via attributes). As we shall see in the right side of Fig. 2, we based our representation on a component vision.

By doing so, we get a definition of a pattern with a clear and flexible structure. Such a structure is already used with success in CBSE (Component-Based Software Engineering). CBSE is a discipline that is known to be reliable in the area of software

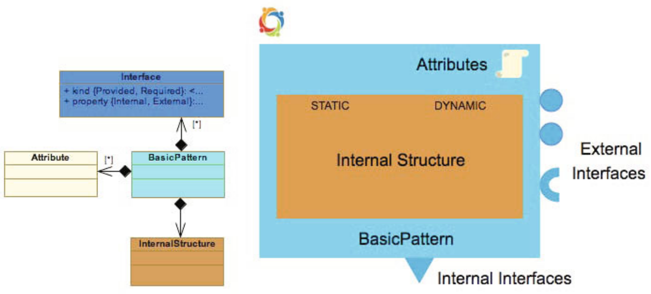

**Fig. 2.** Pattern Fundamental Structure

engineering and that is considered as to be a good solution to optimize software reuse and dynamic evolution while guaranteeing the quality of the software [5]. Moreover, the modularity it enables allows to master the complexity of large systems. The follow paragraphs detail the classes of our meta-model:

**BasicPattern.** This block represents a modular part of a system that encapsulates a solution to a recurrent problem. A *BasicPattern* defines its behavior in terms of provided and required interfaces. As such, a *BasicPattern* serves as a type whose conformance is defined by these provided and required interfaces. One *BasicPattern* may therefore be substituted by another only if the two are type conformant. Larger pieces of a system's functionality may be assembled by reusing patterns as parts in an encompassing pattern or assembly of patterns, and wiring together their required and provided interfaces. A *BasicPattern* is modeled throughout the development life cycle and successively refined into deployment and run-time. A *BasicPattern* may be manifest by one or more artifacts, and in turn, that artifact may be deployed to its execution environment. A deployment specification may define values that parameterize the pattern's execution.

**Interfaces.** A *BasicPattern* possesses provided and required interfaces. A provided interface is implemented by the *BasicPattern* and highlights the services exposed to the environment. A required interface corresponds to services needed by the pattern. Finally, we consider two kinds of interfaces:

- *External interfaces* allow implementing interaction with regard to:
  - *integrate* a *BasicPattern* to an application model. These interfaces are realized by the *BasicPattern*.
  - *compose BasicPatterns* together.
- *Internal interfaces* allow implementing interaction with the application platform. For instance, at a low level, it is possible to define links with software or hardware module for the cryptographic key management.

**Attributes.** This information allows one to classify and to configure pattern's parameters (e.g. S&D parameters). Another issue is expressing the services provided by the *BasicPattern*. Attributes can be used by configuration tools to preset configuration values of a *BasicPattern*.

**InternalStructure.** It describes the implementation of the *BasicPattern*. Thus the InternalStructure can be considered as a white box which exposes the details of the *BasicPattern*.

To make our modeling framework easier and more intuitive, we explore in the subsequent sections only the information related to *Attributes*. That is, we use this information to illustrate the power of our approach at the different level of abstractions. We also detail *Attributes* when we sketch our notable example.

## 2.2 Domain Independent Pattern Model – DIPM

Here, we describe the DIPM (see Fig. 3) as an instance of the PFS meta-model. Its objective is to describe patterns at high level of abstraction without domain properties. Below we outline its principal elements.

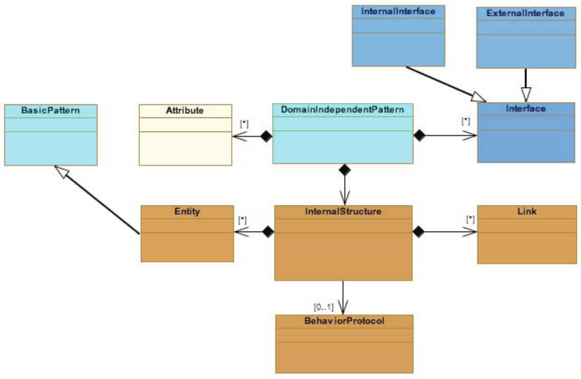

**Fig. 3.** Domain Independent Pattern Model

**DomainIndependentPattern.** This is the representation of a pattern at the model level. It corresponds to an instance of a *BasicPattern*.

**Interfaces.** It is necessary to instantiate the interfaces of a BasicPattern. For instance several kinds of external interfaces can be defined: *Secure Access External Interface*, *Secure Download External Interface*, *Secure Storage External Interface*, etc.

**InternalStructure.** A *Domain Independent Pattern* has an internal structure that describes its implementation. This one is composed of both *Entity* and *Link* which form the structure. A *Behavior Protocol* can be associated to the internal structure in order to describe its behavior.

**Attributes.** One can define several kinds of attributes. For instance, in the context of RCES we propose to address the following standards non-functional properties:
- *Security:* AccessControl, Integrity, Authenticity, Confidentiality, ...;
- *Dependability:* Availability, Reliability, Maintenability, ...;
- *Mechanism:* Cryptographic Algorithm, Integrity Algorithm and Flow Control Protocol,...;
- *RCES:* timeDelay, RamSize, PowerConsumption, ...;
- *SpecificContext:* Ad-hoc Network,...;
- *Quality:* IEEE certification, SIL, ...;
- *BehaviorLanguage:* CSP, ....

### 2.3 Domain Dependent Pattern Model – DDPM

In this subsection, we describe the DDPM (see Fig. 4) which corresponds to a DIPM refinement (model transformation). The motivation of this model is to specify S&D patterns at domain specific application level. In the paper, the authors will not develop all the model but will focus on the specialization of some elements.

**DomainDependentPattern.** It refines the *DomainIndependantPattern*. Several *DomainDependentPatterns* can be built from a same *DomainIndependantPattern*.

**Interfaces.** They are specialized by domain constraints. For example, in Home Control domain, a specific access control can be chosen.

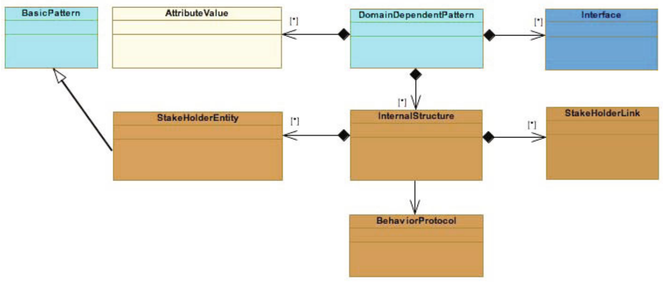

**Fig. 4.** Domain Dependent Pattern Model

**InternalStructure.** It is composed of *StakeHolderEntity* and *StakeHolderLink*. A *behaviorProtcol* can be attached to the *InternalStructure*. For instance, in home control domain, a child, a elderly person, or an Operator subscriber corresponds to *StakeHolderEntity*. Same, *StakeHolderLink* can represent a Wifi connection, HMI (Human Machine Interface).

**AttributeValue.** It details all attributes defined at the DIPM level. For instance, when dealing with mechanism attributes *RC4* and *WEP* can be specified for a cryptographic algorithm and flow control protocol, respectively and so on. Fig. 7 depicts more specific attribute values for the other domain independent attributes.

### 2.4 Summary of the Proposition

To develop patterns, we choose a MDE-based solution. First, we have designed a metamodel in order to define generic patterns with S&D for RCES concern. Then, it is possible to model patterns by following a standard model approach.

In the perspective proposed in this paper, a pattern can be considered at two abstraction levels. The first one is independent of the domain in opposition to the second one which is specific to one concern. At the DIPM (Domain Independent Model) level, it is necessary to define the interfaces, attributes and the internal structure of the pattern. During the specialization of DIPM, all elements defined at the DIPM level can be refined in this phase such as the interfaces, the internal structure and the properties. This step yield a representation of a patterns for a specific domain (DDPM). The approach is highlight by a toy example in the next section.

## 3 Secure Service Discovery Pattern for Home Control

In the following, an example will illustrate the approach point defined in the previous sections. The current trend aims at integrating more intelligence into the homes to increase services to the person. For this purpose, electronics equipments are widely used while providing easy and powerful services. However, to integrate all the services automatically requires a plug and play like system. For this issue, this example aims at providing a pattern for home control domains which provides a secure service discovery. Compared to a usual service discovery, this pattern will use a secure channel in

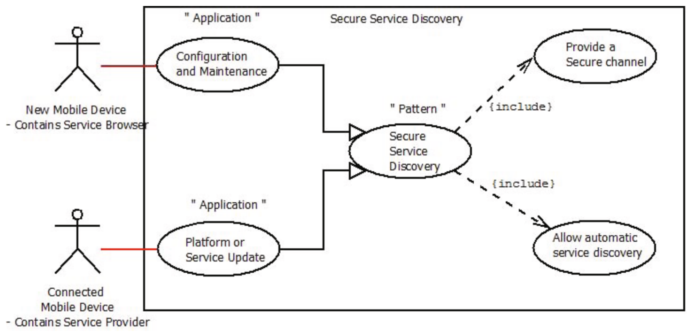
" Application "
Secure Service Discovery
Configuration and Maintenance
Provide a Secure channel
New Mobile Device - Contains Service Browser
" Pattern "
{include}
Secure Service Discovery
{include}
" Application "
Platform or Service Update
Allow automatic service discovery
Connected Mobile Device - Contains Service Provider

**Fig. 5.** Secure Service Discovery Use Case

order to protect all data. Fig. 5 shows two use cases of this pattern: (i) adding a new equipment (ii) updating the current configuration. The Fig. 5 is described in form of UML notations and highlights the interfaces of the pattern in order to support the two main use cases. In the next subsections, the interface and the static internal structure will be pointed out while following the two abstraction levels (i.e., DIPM and DDPM) proposed by the paper.

### 3.1 Representation at DIPM: Person Using a Remote InternetBox

Regarding to the interface, it is necessary to declare some operations which allow the user to check if new implementations exist (i.e., update) and to detect the context (i.e., new equipment). Then, it is necessary to define all properties addressed by the pattern. Fig. 6 illustrates the representation of secure connection pattern at DIPM. At this level, the pattern deals with a person using a remote InternetBox. As mentioned in the previous section, we choose Access control and Crypto algorithm for Security and

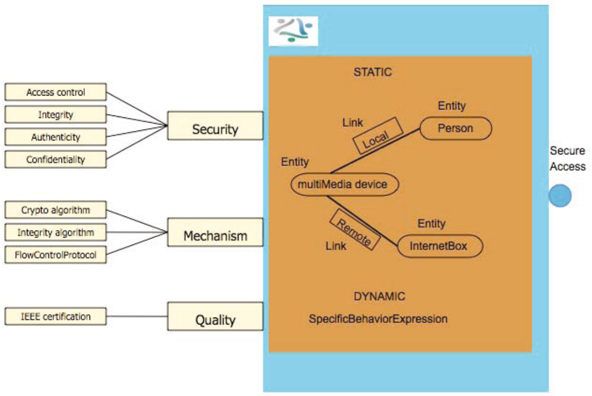

**Fig. 6.** Service Discovery example: Person using a remote InternetBox (DIPM)

Mechanism Attributes, respectively. Regarding the internal structure of the pattern, we consider the following: a person uses a multi-media device which communicates with an internetBox via a Wifi connection.

### 3.2 Specialize a Pattern through the DDPM: Subscriber Using a Remote OperatorBox

The interfaces must be adapted in order to match with the specific communication used in the domain. Regarding to the properties, at the DIPM, we only specify a very generic security property. At this level, it is possible to refine this property by defining the cryptographic mechanism, the length of the keys, etc. Moreover, it is possible to add new properties. For instance, a RCES property can be added like the cryptographic time. Fig. 7 illustrates the representation of secure connection pattern at DDPM. At this level, the pattern deals with an operator subscriber using a 'Operato'Box. For instance, with regard to AttributesValue, we choose OperatorX AC and WEP for Access Control and Flow Control Protocol, respectively. The information expressed by the internal structure of the pattern is the following: an operator subscriber person uses a computer which is connected to a 'Operator'Box.

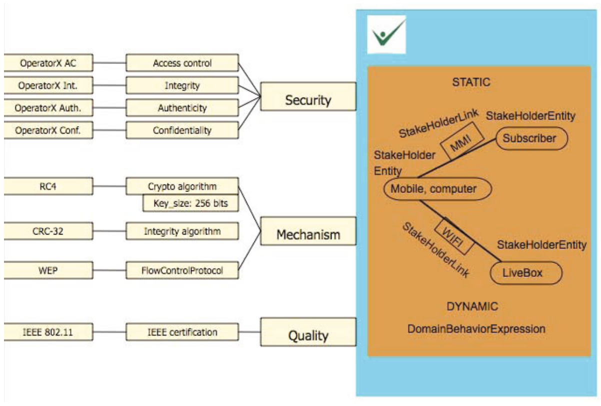

**Fig. 7.** Person using a remote InternetBox (DDPM)

## 4 Related Works

The development of embedded systems has been heavily explored in research areas, ranging from design model level to runtime support level. However they do not give precise support for non-functional properties. Several works have been carried out to deal with such properties but they impose solutions that are programming language-based. Here, we focus on design and modeling techniques for software reliability. Firstly we present a set of connected works including UML extensions, patterns and S&D patterns. Secondly we propose a positioning of the work presented in this paper with regard to the other works.

## 4.1 Connected Works

Many studies have already been done on modeling security in UML. [14] presents an extension UMLsec of UML that enables to express security relevant information within the diagrams in a system specification. UMLsec is defined in form of a UML profile using the UML standard extension mechanisms. [16] presents a modeling language for the model-driven development of secure distributed systems based on UML.Their approach is based on role-based access control with additional support for specifying authorization constraints. SecureUML is a modeling language that defines a vocabulary for annotating UML-based models with information relevant to access control.

The design patterns are usually represented using text-based languages, diagrams with notations such as UML object modeling and most often improved by textual descriptions and examples of code fragments to complete the description. Unfortunately, the use and / or application of pattern can be difficult or inaccurate. In fact, the existing descriptions are not formal definitions and sometimes generate some ambiguities about the exact meaning of the pattern. To give a flavor of the improvement achievable by using specific languages, we look at the pattern formalization problem. The best known frameworks for this problem are *UMLAUT, RBML, LePUS* and *DPML*. *UMLAUT* was proposed by Guennec et al. [2] as an approach that aims to formally model design patterns by proposing extensions to the UML meta model 1.3. In the same way, Kim et al. [6] presented another approach to represent design patterns in a language called *RBML (Role-Based Meta modeling Language)* based on the UML. The RBML is able to capture various design perspectives of patterns such as static structure, interactions, and state-based behavior. This language is based on the meta-modeling design patterns and offer $three$ specifications: Structural, Behavioral and Interactive.

Another issue raised in [10] and [7] is visualization. Eden et al. [10] presented a formal and visual language for specifying design patterns called *LePUS*. With regard to the integration of patterns in software systems, the *DPML (Design Pattern Modeling Language)* [7] allows the incorporation of patterns in UML class models.

Several tentatives exist in the S&D design pattern literature [22,12,23,8,21,19]. They allow to solve very general problems that appear frequently as sub-tasks in the design of systems with security and dependability requirements. These elementary tasks include secure communication, fault tolerance, etc. Particularly, [22] presented a collection of patterns to be used when dealing with application security. The work of [12] reports an empirical experience, about the adopting and eliciting S&D patterns in the Air Traffic Management (ATM) domain, and show the power of using patterns as a guidance to structure the analysis of operational aspects when they are used at the design stage. A survey of approaches to security patterns is proposed in [23].

In addition to the above, the recently completed *FP6 SERENITY* project [1] has introduced a new notion of S&D patterns. SERENITY's S&D patterns are precise specifications of validated security mechanisms, including a precise behavioral description, references to the S&D provided properties, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. Such validated S&D patterns, along with the formal characterization of their behavior and semantic, can also be the basic building blocks for S&D engineering for embedded systems. [20] explains how this can be achieved by using a library of precisely described and formally verified security and dependability (S&D) solutions, i.e., S&D classes, S&D patterns, and S&D integration schemes.

### 4.2 Positioning

The classical approaches (based on Gamma), SERENITY and the work presented in this paper are classified and compared depending of the following criterions: language of representation, kind of structure and levels of abstraction.

For the first kind of approaches [11] , the design patterns are usually represented by diagrams with notations such as UML object, most often accompanied by textual descriptions and examples of code to complete the description. Furthermore their structure is rigid (Context, Structure, Solution, etc.). Unfortunately, the use and / or application of a pattern can be difficult or inaccurate, in effect; the existing descriptions are not formal definitions and sometimes leave some ambiguity about the exact meaning of patterns.

There are some promising and well-proven approaches [9] based on Gamma et al, but nevertheless this kind of works do not allow to reach the high degree of pattern structure flexibility which is required to reach our target.

UMLsec [14] (approach based on modeling security in UML) and our proposal are not in competition but they complement each other by providing different view points to the secure information system.

LePUS [10] is a framework that aims the specification of design patterns. For us, the visualization technique is interesting but the degree of expressivity to design a pattern is too restrictive.

On the other hand, SERENITY [1] proposes a pattern language. Nevertheless the pattern structure is rigid (a pattern is defined as quadruplet) and consequently is not usable to capture specific characteristics of S&D patterns. We can note that SERENITY proposes several levels of abstraction to bridge the gap between abstract solution and implementation. Nevertheless SERENITY does not allow to get a common representation of patterns for several domains. As a side remark, note that our goal is to obtain an even high level abstraction to represent S&D patterns to capture several facets of security and dependability in the different domain of RCES, not an implementation of a specific solution.

Finally, the work presented in this paper proposes an original pattern flexible structure based on a model as its first structural citizen all along the development process. The structure of a pattern here is closest to a component vision (such as defined in CBSE) that allows to use the pattern in RCES on easy way. The different levels of abstraction we propose allow to get a common representation of patterns for several domains.

## 5 Conclusion and Future Work

Security and dependability are not building blocks added to an application at the end of the development life cycle. It is necessary to take into account this concern from the requirement to the integration phases. This paper decides to follow a MDE-based approach to specify patterns. Indeed, MDE solutions allows to meet several concerns around one model while ensuring coherence between all businesses. For this, we need to develop a new pattern development technique supporting flexibility in order to meet S&D needs. The proposed approach is structured in 3-layer architecture. The first one corresponds to a metamodel which define a generic structure of patterns. Then, two other layers are an instance of the metamodel. The two last levels allows us to integrate domain specific features at the end of the process.

The next step is primarily the definition of a language to specify S&D and RCES properties within the pattern. Existing work such as MARTE standard (Modeling and Analysis of Real-Time Embedded systems) [17] and the characterization given in [4] can be good candidate as the basis for this type of concern. Then, we will focus on the integration of all the presented results in a more global process. The challenges of the integration include for instance the complete pattern development life cycle (i.e., create, update, store patterns). All patterns are stored in a repository. Thanks to it, it is possible to find a pattern regarding to S&D criterion. Patterns and the application are formally validated regarding to S&D properties. Another objective for the near future is to provide guidelines during the pattern development and the application development (i.e., help to choose the good pattern) in order to enhance their integration during a software engineering process.

## References

1. Serenity, system engineering for security & dependability (2006), http://www.serenity-project.org
2. Le Guennec, A., Sunyé, G., Jézéquel, J.-M.: Precise Modeling of Design Patterns. In: Evans, A., Caskurlu, B., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 482–496. Springer, Heidelberg (2000)
3. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Center for Environmental Structure Series, vol. 2. Oxford University Press, New York (1977)
4. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1, 11–33 (2004)
5. Brown, A.W., Wallnau, K.C.: The current state of CBSE. IEEE Software 15(5), 37–46 (1998)
6. Ghosh, S., Kim, D.-K., France, R., Song, E.: A uml-based metamodeling language to specify design patterns. In: Patterns, Proc. Workshop Software Model Eng. (WiSME) with Unified Modeling Language Conf. 2004 (2004)
7. Grundy, J., Mapelsden, D., Hosking, J.: Design pattern modelling and instantiation using dpml. In: CRPIT 2002: Proceedings of the Fortieth International Conference on Tools Pacific, pp. 3–11. Australian Computer Society, Inc. (2002)
8. Daniels, F.: The reliable hybrid pattern: A generalized software fault tolerant design pattern. In: PLOP 1997 (1997)
9. Douglass, B.P.: Real-time UML: Developing Efficient Objects for Embedded Systems. Addison-Wesley (1998)
10. Gasparis, E., Nicholson, J., Eden, A.H.: LePUS3: An Object-Oriented Design Description Language. In: Stapleton, G., Howse, J., Lee, J. (eds.) Diagrams 2008. LNCS (LNAI), vol. 5223, pp. 364–367. Springer, Heidelberg (2008)
11. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)

12. Di Giacomo, V., et al.: Using security and dependability patterns for reaction processes, pp. 315–319. IEEE Computer Society (2008)
13. Henzinger, T.A.: Two challenges in embedded systems design: Predictability and robustness. Philosophical Transactions of the Royal Society A 366, 3727–3736 (2008)
14. Jürjens, J.: Umlsec: Extending uml for Secure Systems Development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)
15. Kopetz, H.: The complexity challenge in embedded system design. In: ISORC, pp. 3–12 (2008)
16. Lodderstedt, T., Basin, D.A., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
17. OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems,beta 2 (June 2008), http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf
18. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: Design challenges. ACM Trans. Embed. Comput. Syst. 3(3), 461–491 (2004)
19. Schumacher, M.: Security Engineering with Patterns. LNCS, vol. 2754. Springer, Heidelberg (2003)
20. Serrano, D., Mana, A., Sotirious, A.-D.: Towards precise and certified security patterns. In: Proceedings of 2nd International Workshop on Secure Systems Methodologies Using Patterns (Spattern 2008), pp. 287–291. IEEE Computer Society (September 2008)
21. Tichy, M., et al.: Design of self-managing dependable systems with uml and fault tolerance patterns. In: WOSS 2004: Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems, pp. 105–109. ACM (2004)
22. Yoder, J., Barcalow, J.: Architectural patterns for enabling application security. In: Conference on Pattern Languages of Programs, PLoP 1997 (1998)
23. Yoshioka, N., Washizaki, H., Maruyama, K.: A survey of security patterns. Progress in Informatics, 35–47 (2008)
24. Zurawski, R.: Embedded systems. In: Embedded Systems Handbook. CRC Press Inc. (2005)

# Modeling Ad-Hoc Collaboration for Automated Process Support

Komlan Akpédjé Kedji[1,2], Bernard Coulette[1], Redouane Lbath[1], and Mahmoud Nassar[2]

[1] Institut de Recherche en Informatique de Toulouse, Toulouse, France
{kedji,coulette,lbath}@univ-tlse2.fr
[2] Universite Mohamed V Souissi, Rabat, Morocco
nassar@ensias.ma

**Abstract.** Understanding and supporting collaboration is a major concern in any collective endeavor. In software engineering, planning collaboration is difficult and often occurs rather ad-hoc. However, many proposals for the description and support of collaboration assume necessary information is readily available and already integrated in process models. We believe practitioners should be given the ability to represent their evolving understanding about collaboration and should be supported by tools using such representation. We developed an extension to the Software & System Process Engineering Meta-Model (SPEM), which introduces concepts needed to represent precise and dynamic ad-hoc collaboration setups. We also describe tool support and present a proof-of-concept based on a fictitious but realistic application example.

**Keywords:** process planning, software quality, collaboration.

## 1 Introduction

Collaboration can be defined as a collective effort to construct a shared understanding of a problem and its solution [1], and is a pervasive concern in any collective endeavor. This effort can be hindered when communication, recognition, trust, etc. incur too much cognitive overhead. In software engineering, these problems are further complicated by frequent worldwide distribution of teams, the amount of knowledge sharing required, communication among people with different expertise, and high complexity and interdependency of artifacts [2].

Ad-hoc collaboration is prompted by a new issue or new considerations in an existing issue, while planned collaboration happens because it was on the agenda. When collaboration is needed because of the requirements-imposed amount of work, which can be somewhat predicted, prior planning can help to sort things out. However, everyday collaboration can be prompted but participants' involvement in the same work item or their interest (as users or editors) in the same work product. Such situations are context-dependant, and each one needs a tailored solution. It has been shown that while there can be a certain amount of prior planning, collaboration, especially in software engineering, is significantly ad-hoc [3,4] (as much as 69% of all collaborative work in a field study [4]).

A collaboration support tool should use concepts familiar to its users, so as to not increase the cognitive overhead they are fighting with, and provide a frictionless surface for development [5]. However, concepts used in popular generic process formalisms (role, task, product) are not fine-grained enough to describe ad-hoc collaboration, especially on the detailled project plan level:

– Role: how do we describe interactions between people playing the same role, but implementing two different components with dependencies between them, when the very need for two people surfaces only in the middle of the project?
– Task: when we find out that more than one person is needed for a task, how can we, for example, specify work dependency, when we lack a concept to represent work done by a specific person in the context of a task?
– Product: how do we describe (temporary) agreements for merging changes to produce an official version of a work product, from copies local to participants, while lacking the ability to identify each of those copies?

Admittedly, existing abstractions try to make it easy to grasp huge process models, and promote the reuse of process models by stripping them of project specific details. However, in everyday collaboration, participants are only interested in a small, specific part of the process model. Before thinking about reuse, they need immediate support for their current endeavor.

We propose to postpone the description of ad-hoc collaboration, so that the creative collaboration setups practitioners create to answer their precise needs can be taken into account (which helps maintain the connection with reality [6]), using the more appropriate concepts that we introduce. This description precises the standard description (based on roles, products, and tasks) which remains generic and reusable.

Process quality and product quality have been repeatedly confirmed as linked by researchers and industrials [7,8,9]. Formalizing software processes as models clarifies project concerns for all stakeholders, and also provides guidance on task execution. However, process models are usually considered as a somewhat rigid specification that must be enforced. We consider process models as living descriptions of decisions about collaboration. Quality improvement is then realized by the analysis and eventual automation of dynamic, day to day decisions made by practitioners and represented in the process model. This naturally results in a feedback loop, where the process models guides short-term actions, and those actions in turn help improve the process model. Additionally, the accuracy of such representation enables a better evaluation of past decisions, which is necessary for process improvement.

The rest of this paper is organized as follows. Section 2 presents some related works, and section 3 formalizes our contribution as a metamodel. Section 4 describes typical ad-hoc updates to process models, and section 5 shows how models based on our metamodel are to be created, updated, and exploited by tools.

## 2 Related Works

Existing contributions studied the difficulty of formalizing knowledge about work practices, and flexible instantiation of process models. Enhancing the precision of process metamodels, and giving practitioners the ability to improve process models with their evolving understanding have also been investigated.

In [10], M. Polanyi et al., discuss how practitioners rely on implicit knowledge that is really hard if not impossible to formalize, summed up as "we know more than we can tell". It is even harder to formalize knowledge in the abstract, outside of any practical situation. Therefore, our goal is to give practitioners the ability to represent their *ongoing* interactions, thus departing from the approaches that seek to embed fixed setups, once for all, in process models.

Killisperger et al., developed a framework for flexible process instantiation [11]. The goal is to assist step-by-step tailoring and instantiation of generic and complex process models. The progressive approach is shared by our proposal, but the goal of [11] is to make sure tailoring and instantiation respect certain pre-defined rules (syntax and organizational rules), while we focus on collaboration.

There have been other efforts to extend SPEM. For example, [12] defines a formalism based on Petri nets geared towards precise definition of MDE (Model Driven Engineering) process models and process execution tracking. It however ignores resource allocation and roles definition, and concentrates on process steps (described with links, models, flow, resource, etc.) This formalism is more detailed than SPEM, but follows the same "define and execute" approach.

Witshel et al. [13] identified the need to make business process execution more flexible. They start from the insight that while offering valuable context information traditional business process modeling approaches are too rigid to capture the actual way processes are executed. The paper explores how practitioners can use their knowledge to enhance business process models, by leaving semi-structured comments on business activities. Our contribution, while focused on software processes (where deviations are also frequent), shares this feedback approach, where information from actual execution enriches the process model.

In [6], Grudin et al. proposed a methodology for the design of collaborative environment which is halfway between the top-down approach of put-all-the-knowledge-in-at-the-beginning and the bottom up approach of just-provide-an-empty-framework. Witshel et al. [13] used this approach to design a "task pattern" formalism (task patterns are defined as "abstractions of tasks that provide information and experience that is generally relevant for the task execution"). Task patterns are instantiated by assigning real persons to the positions defined in the task pattern. A user can enhance a task pattern while executing it. The enhancements are stored locally, reused automatically for subsequent instances of the task pattern, and can be published so others can use it.

In [14], Alegria et al. use three perspectives (or views: role, task, work product) to visualize process models, and discover their shortcomings. The usefulness of visualisation is most manifest for rich models, with a lot of details about how the process is actually being executed. This makes it a perfect fit for our approach, as the following sections will demonstrate.

## 3 The CM_SPEM Metamodel

Our conceptualization is presented as a metamodel, CM_SPEM (Collaborative Model-based Software & Systems Process Engineering Metamodel), that extends SPEM [15] with ad-hoc collaboration description capabilities.

The SPEM metamodel is an industry standard introduced by the OMG (Object Management Group). Its goal is to "accommodate a large range of processes, and not to exclude them by having too many features or constraints" [15]. SPEM is structured around the notion that process elements can be defined once, and reused as needed in process models. Mapping to existing process formalisms like RUP (Rational Unified Process [16]) are provided, to demonstrate that SPEM fulfills its goal as a base lingua franca for process modeling.

The main concepts our metamodel borrows from SPEM are RoleUse (from SPEM::ProcessStructure), ProductUse (from SPEM::ProcessStructure) and TaskUse (from SPEM::ProcessWithMethods). The main concepts introduced are Actor, ActorSpecificTask, and ActorSpecificArtifact. The other concepts are relationships between the newly introduced concepts. Our additions to SPEM are isolated in a "Collaboration Structure" package. (Figure 1). The OCL (Object Constraint Language) [17] rules which specify the static semantics of the metamodel as well-formedness rules have not been included for brevity.

To illustrate our metamodel, we consider a fictitious by realistic project concerned with the development of a complex ticket reservation system (later referred to as "CTRS project"). A continuous deployment strategy is used, that is, new features are regularly pushed into production, as they are requested by the client and implemented by the team. The team in charge of the CTRS project is composed of the following participants:

– Bob, the designer. Bob designs and writes architecture description models. These models are used to generate interfaces and data conversion code.
– Alice, the integration manager. Alice's role is to decide when features and fixes are ready for production, and merge them while making sure the result is functional and reasonably bug-free.
– Fred, the deployment manager. Fred deploys the project to production, monitors execution, and reports errors back to developers.
– Karl and Mike, developers. They mostly write code which implements performance sensitive functionalities and integration with legacy systems.
– Tracy, developer. She writes integration tests for the system under development.

Most of the features of the system being implemented are original, which means a lot of experimentation is done. The whole team is also not collocated. The team uses a distributed version control so as to exploit the flexible branching and local history manipulation capabilities needed in such situation.

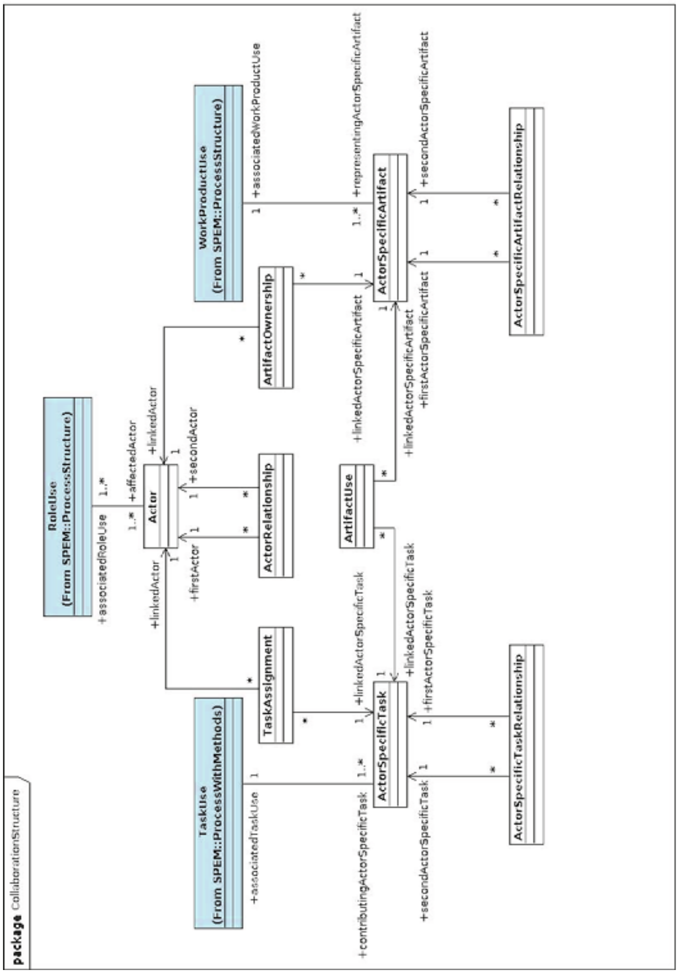
package CollaborationStructure
TaskUse (From SPEM::ProcessWithMethods)
RoleUse (From SPEM::ProcessStructure)
WorkProductUse (From SPEM::ProcessStructure)
Actor
ActorRelationship
ArtifactOwnership
TaskAssignment
ArtifactUse
ActorSpecificTask
ActorSpecificArtifact
ActorSpecificTaskRelationship
ActorSpecificArtifactRelationship
+associatedTaskUse
+contributingActorSpecificTask
+secondActorSpecificTask
+firstActorSpecificTask
+linkedActorSpecificTask
+linkedActor
+firstActor
+secondActor
+affectedActor
+associatedRoleUse
+linkedActorSpecificArtifact
+firstActorSpecificArtifact
+secondActorSpecificArtifact
+representingActorSpecificArtifact
+associatedWorkProductUse

**Fig. 1.** The Collaboration Structure package (shaded concepts come from SPEM)

### 3.1 Structural Concepts

**Central Concepts.** The people participating in the project, and their respective roles are: Bob (Architect), Alice (Integration manager), Fred (System administrator), Karl (Developer), Mike (Developer), Tracy (Developer). In CM_SPEM, people are called "actors". An actor unambiguously identifies a specific human participant in a project. It is linked to instances of RoleUse (SPEM::ProcessStructure) to specify that an actor plays the corresponding roles.

The tasks (SPEM TaskUse) involved are: "Edit architecture description model" (Bob), "Implement components" (Karl and Mike), "Integrate and deploy" (Alice), and “Write tests” (Tracy). For each task, we can specify how the different actors participate in them, with “actor specific tasks”. In the special case of the “Implement components” task for example, two actor specific tasks are created, as two actors are assigned to it. An ActorSpecificTask is a unit of work done by a specific actor, towards the execution of a TaskUse. An ActorSpecificTask is linked to exactly one instance of TaskUse (SPEM::ProcessWithMethods) to indicate that it contributes to the realization of the corresponding TaskUse.

For each work product, the copy in each developer workspace is an “actor specific artifact”. An ActorSpecificArtifact is a physical occurrence of a WorkProductUse, in the personal workspace of a specific actor. This is the personal copy of the actor, and is manipulated only by him/her. An ActorSpecificArtifact is linked to exactly one instance of WorkProductUse (SPEM::ProcessStructure), and specifies that the actor specific artifact represents the corresponding work product. In other words, a work product is the collection of all actor specific artifacts that represent it. One copy is usually designated as the reference version, and normally belongs to the integration manager or to the deployment manager in software engineering projects.

Actor, ActorSpecificTask, and ActorSpecificArtifact are the central concepts in CM_SPEM, and each of them extend ExtensibleElement (from SPEM::Core).

**Relationships.** In the considered development setup, each feature is first prototyped in a developer’s local (software configuration management) repository, then pulled into the integration manager’s repository, before going into the official repository, which is used in production. Developers bring their local repositories up to date by pulling artifacts from the official repository. An initial set of relationships can be introduced (Figure 2), to reflect this setup. These relationships are essentially actor relationships:

– “Pulls from”, which specifies where an actor pulls (retrieves) contributions from by default.
– “Pushes to” which specifies where an actor sends contributions to by default.

Relationships encode knowledge about collaboration with links between concepts. All relationships are subclasses of SPEM::Core::ExtensibleElement (which allow them to have user-defined qualifications) and SPEM::Core::BreakdownElement (which allows them to be nested by SPEM containers like Activity).

User-defined qualifications (applicable to any subclass of SPEM::Core:: ExtensibleElement) are additional, detailed semantics that a modeler can apply to SPEM model elements. This SPEM mechanism is heavily used in CM_SPEM to allow practitioners to augment relationships defined bellow with the precise semantics needed for their use. For example, when defining a relationship between two actors, one can specify that this relationship means that one actor reports to the other in the context of a specific task.

The following set of relationships are used to relate the central concepts one to another: TaskAssignment which relates an Actor to an ActorSpecificTask assigned to him/her, ArtifactOwnership which relates an Actor to an ActorSpecificArtifact that belongs to his/her workspace, and ArtifactUse which relates an

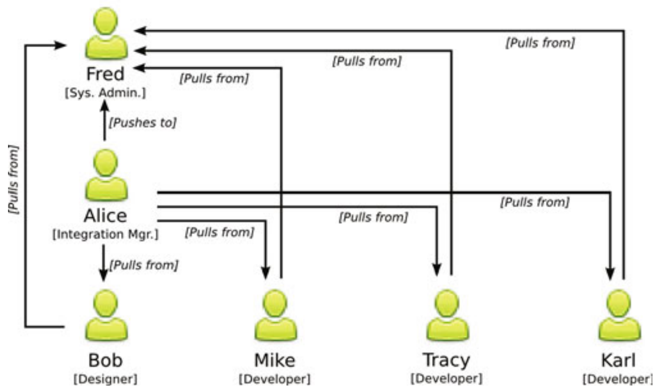

**Fig. 2.** Actor relationships in the initial model

ActorSpecificTask to an ActorSpecificArtifact that is manipulated when carrying the task out.

To describe the interactions of a couple of instances of one of the central concepts (Actor, ActorSpecificTask, or ActorSpecificArtifact), the following relationships are used:

– ActorRelationship, which is used to describe a couple of actors. As an extensible element, an ActorRelationship can be qualified by an instance of ActorRelationshipKind (which inherits from SPEM::Core::Kind), using the SPEM user-defined extension mechanism. An ActorRelationship can be used to state that an actor A reports to actor B in a certain task, so that all contributions made by A, in the context of that task, are by default sent to B (when executing the "push" command in a version control system for example).
– ActorSpecificTaskRelationship, which is used to describe a couple of ActorSpecificTasks. The extension mechanism is similar to the one used for ActorRelationship. An ActorSpecificTaskRelationship can for example specify temporal dependencies between the works done by two different actors in the context of the same TaskUse.
– ActorSpecificArtifactRelationship, which is used to describe a couple of ActorSpecificArtifacts. The extension mechanism is also similar to the one used for ActorRelationship. An ActorSpecificArtifactRelationship can for example specify that an artifact A is the reference version of an artifact B (A and B represent the same work product).

### 3.2 Behavioral Concepts

Behavioral modeling in CM_SPEM relates to how models can evolve over time, and how this evolution is accounted for. In other words, behavioral modeling is concerned with how changes to the process model are detected, transmitted, and reacted to. Besides the state-machine based behavioral model [18] that has already been developed to handle MDE aspects (like transformations), we propose, for collaboration support, an event-based behavioral model (SPEM allows connecting to different behavioral models). We also introduce the concepts of ToolUse, which allows tools (which assist practitioners) to be represented in process models. This gives them the capability to generate and subscribe to events as any other model elements, as described below.

In the CTRS project for example, the “Pulls from” and “Pushes to” relationships introduced earlier are supported by a fictitious tool, which we will name “DVCS Configuration tool”, which configures the version control system based on the setup described in the model. This tool listens for “PullsFrom relationship creation” and “PushesTo relationship creation” events. Whenever such relationships are added to the model, the tool is notified. In this specific case, it can setup remotes, branch tracking, default merge strategies, etc.

An event is a single occurrence of something of interest. Events have parameters, whose values describe the event. This concept is a reuse of the Event concept defined for state machines in UML. Event classes are used to categorize events. Events are raised (that is, generated) by event sources (any CM_SPEM model element which can trigger events), and received by event handlers (any CM_SPEM model element which can receive events an react to them). To be able to receive an event, a handler must subscribe to an event class prior to the occurrence of the event.

## 4 Dynamic Updates to a Process Model: Representation in CM_SPEM

The essential feature of CM_SPEM is its ability to cope with additional decisions that are made in the project as new information becomes available. This section discusses a few situations derived from the CTRS project presented in the previous section, and describes how they could be handled.

### 4.1 A New Member Is Added to the Team

The input validation code is found to be repetitive, and the team decides to convert part of the hand-written code to models, backed by home-grown code-generators. A new participant, Sue, a designer, is added to the team to assist Bob in the task. Sue is thus instructed to send her contributions to Bob, who now acts as a sub-system integrator. The implication for the DVCS configuration is that the “Pulls from” relationship, which should have been created between Alice and Sue is instead between Bob and Sue. This is illustrated in Figure 3.

### 4.2 A Serious Bug Is Found, and New Features Are Frozen Until Resolution

In the course of the project, a bug is discovered by some new integration test written by Tracy. The bug proved to be a serious one, requiring a tight

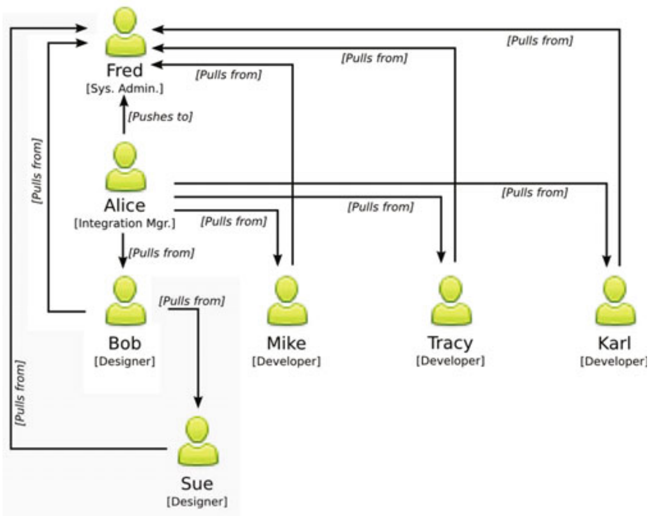

**Fig. 3.** Actor relationships after the integration of Sue in the team

collaboration between Bob (the designer), and all the developers (Karl, Mike, and Tracy).

Karl is coordinating the effort in this particular bug-hunting. The team soon realizes that a different setup is being used for the purpose of tracking and solving this particular bug. As the effort is taking several days, the team decides to introduce the new, temporary setup in the model, so as to have adequate tool support.

A new version control branch is created for the purpose of developing a fix. Karl acts as the integration manager on this branch, with respect to other developers, until the bug is fixed (Figure 4).

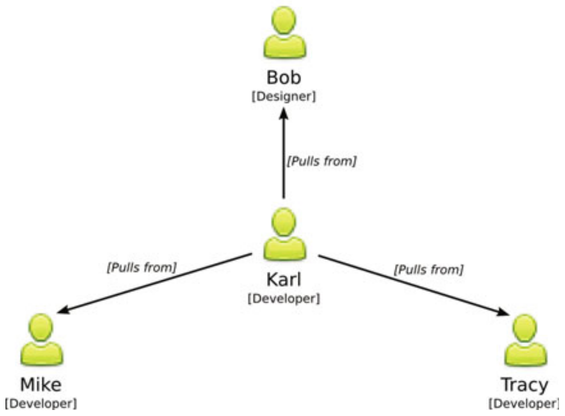

**Fig. 4.** Actor relationships for the work done on the bug fixing branch

## 5 Process Modeling and Enactment Using CM_SPEM

This section provides guidelines about how CM_SPEM models are built and used. The ultimate goal of this work is to ease the development of tools that improve quality by assisting developers in their collaborative endeavors. To this end, there should be a practical way of creating process models with collaboration information, continually enriching process models with new information on collaboration, and using information extracted from process models to enhance quality by automating boilerplate or recurring concerns.

### 5.1 Creating CM_SPEM Process Models

To edit process models, we have defined a graphical notation which extends the graphical representation in SPEM. A graphical editor, which supports this notation, has been developed on top of TOPCASED (Toolkit in Open Source for Critical Applications & Systems Development) [19]. Figure 5 shows a screenshot of the editor with a sample model loaded.

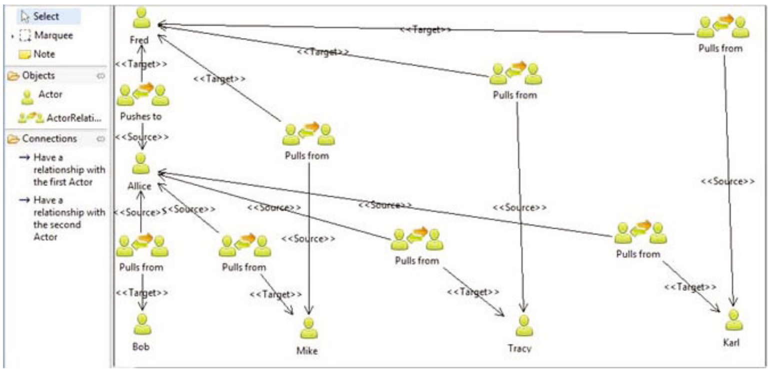

**Fig. 5.** TOPCASED-based editor for CM_SPEM

### 5.2 Modifying CM_SPEM Process Models

When using CM_SPEM, a process model conforming to SPEM acts as a baseline. New elements, specific to CM_SPEM, are introduced as needed, and linked with their SPEM counterparts (actors are linked to role uses, actor specific tasks to task uses, and actor specific artifacts to work products), and relations which capture the collaboration setup are introduced between the concepts.

As elements are introduced in a running (in the sense that it is already being exploited by some tool) process model, care must be taken so that the new elements do not conflict with existing elements. This concern can be minimized by considering all relationships as declarative data about a collaborative setup. In other words, a relationship should always say "how things are done", instead of telling "how things should be done", or "how work is supported by tools". This lowers the coupling between process models and how they are initially used.

### 5.3 Extracting Information from CM_SPEM Process Models

Any tool which supports collaboration can query the process model whenever they need to take action based on its content. This query interface is the CM_SPEM API (Application Programming Interface), and acts as a bridge between CM_SPEM models and CM_SPEM-based tools.

Besides being able to directly extract data from the process model, tools are also represented in process models, which allows them to receive event notifications as any other model element, and react to them. The example discussed in Section 4 is an illustration of how this capability could be used. At the time of writing, only the TOPCASED-based editor for CM_SPEM models is available. The query API and sample tools are not yet ready.

## 6 Conclusion

This work introduced CM_SPEM, a SPEM extension for the description of ad-hoc collaborative work. Our guiding principle is that collaboration should be conceptualized using the ideas most familiar to people collaborating. We hope that the additional automation opportunities provided by the detailed process models based on CM_SPEM will help reduce repetition, and lower the cognitive load on practitioners. This translates into a better quality of the end product.

An editor is already available for CM_SPEM process models (developed with TOPCASED, with OCL-based model checking functionality). We plan to add other diagram types or visualizations to ease understanding, define a complete query API for CM_SPEM process models, and develop an ecosystem of tools which support collaboration based on information extracted from process models.

Our effort is part of a research project where an industrial case study in avionics will be soon described with CM_SPEM, as a real-world validation. This will also help us evaluate the additional effort needed to maintain typical CM_SPEM models, and the overall impact on productivity. One possibility we are investigating for reducing such effort is to partially automate the inclusion of concepts and relationships in models by mining the log files of developer tools.

**Acknowledgment.** This work is conducted in the context of the Galaxy Project, funded by ANR France. The project explores collaborative development of complex systems using heterogeneous development environments and following the MDE approach.

## References

1. Roschelle, J., Teasley, S.: The construction of shared knowledge in collaborative problem solving. NATO ASI Series F Computer and Systems Sciences 128, 69–69 (1994)
2. Mistrík, I., Grundy, J., van der Hoek, A., Whitehead, J.: Collaborative Software Engineering: Challenges and Prospects. Springer, Heidelberg (2010)

3. Cherry, S., Robillard, P.: The social side of software engineering–A real ad hoc collaboration network. International Journal of Human-Computer Studies 66(7), 495–505 (2008)
4. Robillard, P., Robillard, M.: Types of collaborative work in software engineering. Journal of Systems and Software 53(3), 219–224 (2000)
5. Booch, G., Brown, A.: Collaborative development environments. Advances in Computers 59, 1–27 (2003)
6. Grudin, J., McCall, R., Ostwald, J., Shipman, F.: Seeding, Evolutionary Growth, and Reseeding: The Incremental Development of Collaborative Design Environments. Coordination Theory and Collaboration Technology, 447 (2001)
7. Harter, D.E., Slaughter, S.A.: Process maturity and software quality: a field study. In: Proceedings of the Twenty First International Conference on Information Systems, ICIS 2000, pp. 407–411. Association for Information Systems, Atlanta (2000)
8. Rubin, H.A.: Software process maturity: measuring its impact on productivity and quality. In: Proceedings of the 15th International Conference on Software Engineering, ICSE 1993, pp. 468–476. IEEE Computer Society Press, Los Alamitos (1993)
9. Ashrafi, N.: The impact of software process improvement on quality: in theory and practice. Information & Management 40(7), 677–690 (2003)
10. Polanyi, M., Sen, A.: The tacit dimension. University of Chicago Press (2009)
11. Killisperger, P., Stumptner, M., Peters, G., Grossmann, G., Stückl, T.: A Framework for the Flexible Instantiation of Large Scale Software Process Tailoring. In: Münch, J., Yang, Y., Schäfer, W. (eds.) ICSP 2010. LNCS, vol. 6195, pp. 100–111. Springer, Heidelberg (2010)
12. Porres, I., Valiente, M.C.: Process Definition and Project Tracking in Model Driven Engineering. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 127–141. Springer, Heidelberg (2006)
13. Witschel, H.F., Hu, B., Riss, U.V., Thönssen, B., Brun, R., Martin, A., Hinkelmann, K.: A Collaborative Approach to Maturing Process-Related Knowledge. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 343–358. Springer, Heidelberg (2010)
14. Alegría, J.A.H., Lagos, A., Bergel, A., Bastarrica, M.C.: Software Process Model Blueprints. In: Münch, J., Yang, Y., Schäfer, W. (eds.) ICSP 2010. LNCS, vol. 6195, pp. 273–284. Springer, Heidelberg (2010)
15. OMG: Software process engineering metamodel, version 2.0 (2007), http://www.omg.org/spec/SPEM/2.0/
16. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)
17. OMG: Object constraint language (2010), http://www.omg.org/spec/OCL/2.2/
18. Diaw, S., Lbath, R., Coulette, B.: Specification and implementation of SPEM4MDE, a metamodel for MDE software processes. In: International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), Miami - USA, pp. 646–653. Knowledge Systems Institute (2011)
19. Team, T.T.: Toolkit in open source for critical applications and system development (2011), http://www.topcased.org/

# Quality Needs Structure: Industrial Experiences in Systematically Defining Software Security Requirements

Christian Frühwirth[1] and Richard Mordinyi[2]

[1] Software Business Lab, BIT Research Center
Aaalto University – School of Science and Technology
Espoo, Finland
christian.fruehwirth@tkk.fi

[2] Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems", Vienna University of Technology
1040 Vienna, Austria
richard.mordinyi@tuwien.ac.at

**Abstract.** Successful, quality software projects need to be able to rely on a sufficient level of security in order to manage the technical, legal and business risks that arise from distributed development. The definition of a 'sufficient' level of security however, is typically only captured in implicit requirements that are rarely gathered in a methodological way. Such an unstructured approach makes the work of quality managers incredibly difficult and often forces developers to unwillingly operate in an unclear/undefined security state throughout the project. Ideally, security requirements are elicited in methodological manner enabling a structured storage, retrieval, or checking of requirements. In this paper we report on the experiences of applying a structured requirements elicitation method and list a set of gathered reference security requirements. The reported experiences were gathered in an industrial setting using the open source platform OpenCIT in cooperation with industry partners. The output of this work enables security and quality conscious stakeholders in a software project to draw from our experiences and evaluate against a reference base line.

**Keywords:** Distributed Software Engineering, Security Requirements.

## 1 Introduction

Software quality and security are two closely connected issues in global software engineering practices. Wherever a software project requires it's stakeholders to work together, in distributed multi-engineering environments, it is essential for maintaining product quality that the information exchange between the engineers' specialized development tools is 1.) integrated 2.) reliable and 3.) secure. [1] [2]. A promising approach to address these quality challenges was implemented in the Enterprise Service Bus (ESB) [4]. The ESB facilitates the service-oriented architecture (SOA) [3] paradigm, which is suitable to bridge most of the technical and semantic gaps between engineering systems and tools. The ESB has been successfully applied as agile integration platform for services in distributed, heterogeneous business software environments [4].

Previous work has addressed the quality challenges in technical integration of this type of service bus, like message exchange between semantically different engineering tools [12,13]. This paper set out to add security quality considerations as new aspect to the existing work.

The motivation to address security in this context comes from the experience that successful industrial software projects need to be able rely on a sufficient level of security in order to manage the technical (e.g., malware infections), legal (e.g., regulatory compliance), and business (e.g., damage liabilities) risks which arise from distributed development. In today's practice, such levels of security are often captured in merely implicit requirements, and rarely covered in a methodological way. This unstructured approach makes the work of quality managers incredibly difficult and often forces developers to unwillingly operate in an unclear/undefined security state throughout the project. In a best practice case, security requirements would be elicited in a methodological manner that enables the structured storage and retrieval of requirements throughout the lifetime of a software project. In our experience, software project and quality managers are well aware of these best practices, but refrain from their use due to a lack of personal or reported application experiences.

Our work addresses this problem by 1.) exemplifying the methodological elicitation of key security requirements for a collaborative software engineering environment and 2.) reporting on the gained experiences. We applied a method proposed by Mellado [10] to conduct the methodological requirements elicitation. Mellado approach was chosen for its structured format and easy extendibility. The reported experiences were gathered in an industrial setting using the open source platform OpenCIT [5] in cooperation with partners from the company logi.cals .

In addition to providing an application experiences, this work further contributes to the discourse on software quality and security requirements by 1.) providing a reference set of key security requirements for future quality management efforts, 2.) demonstrating the practical application of misuse cases in combination with Mellado's [7] approach, and 3.) proposing a structured requirements assignment matrix to improve the light weight categorization and handling of security requirements by quality managers and application developers.

This work places its focus on the elicitation of security requirements because it is one of the first in a serious of steps towards improved software product quality. Equally important activities however are needed in the coding and testing phases of a product. We consider these phases outside of the scope of this paper but plan to include them in future work on the topic.

The remainder of this paper is structured as follows: Section 2 summarizes related work on aspects of distributed software engineering, elicitation techniques, and security requirements. Section 3 describes the findings of applying a standard elicitation concept on CI&T. Finally, section 4 summarizes our gained experiences while section 5 concludes the paper and identifies further work.

## 2 Related Work

This section summarizes related work on platforms regarding distributed software engineering, security requirements in distributed development and methods for security elicitation in the literature.

### 2.1 Software and System Integration for Distributed Software Engineering

Biffl and Schatten proposed a platform called Engineering Service Bus (EngSB), which integrates heterogeneous engineering systems and tools as well as different steps in the software development lifecycle [6, 7]. Figure 1 gives an overview of the EngSB platform and its core components.

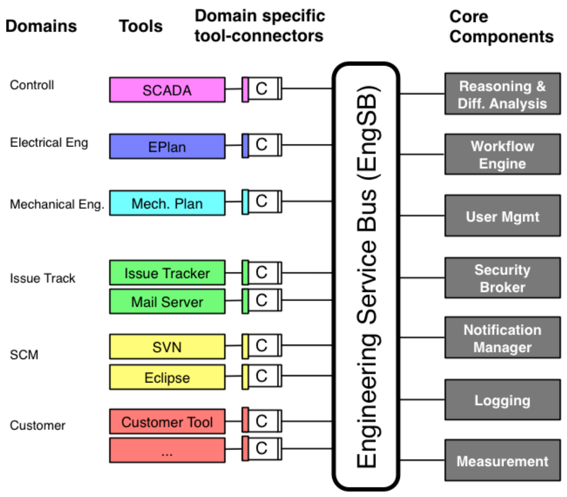

**Fig. 1.** Engineering Service Bus with connected software engineering tools

The EngSB platform introduces the concept of tool types (i.e. tool domains) that provide different tools with common interfaces for solving a common problem, independent of the specific tool instance used. Without then EngSB or an EngSB like system, the communication between different tools is often handled in a semi-manual manner, using spreadsheets, custom scripts, or the like. These traditional approaches are heavily error prone and thus a serious issue for quality conscious project managers.

The EngSB platform is used as basis for numerous industry applications. Our work relates to the the Open Continuous Integration and Test (OpenCIT) server [5] prototype which is based on the EngSB platform and can be used to automatically build, test and deploy software projects. It integrates common tools like SCM, issue tracker, notification server, policy management, etc. relevant for quality control in software engineering projects. The industry project that supported this work chose

OpenCIT over other CI&T tools, like Hudson, for its ability to exchange tool instances [11] and integration of a tool domain concept. More Information on the tool domain concept is available at [11,1].

## 2.2 Security Requirements in Distributed Development

A software project requires its engineering assets to maintain an acceptable level of availability, integrity and confidentiality throughout the project lifecycle. Specifying what is acceptable in a given context is to define security quality requirements. In globally distributed engineering environments however, this specification process can be complex, as the 'context' of an action and resources becomes harder to pin down. In distributed software projects, the project stakeholders operate with large numbers of shared resources that make it difficult to assign ownership and requirements to individual engineering assets [3]. Traditionally security decisions are based on the factors *User/Role -> Action -> Resource*. In a distributed environment however, dynamic factors, such as *previous action (pre-conditions), post-conditions and dependencies* have to be taken into account. Authors like [7,9] discuss these issues in extensive detail. In the scope of this work, we focus on the challenges of eliciting security requirements for the engineering tools that enable a distributed software engineering project. Most engineering tools that are used in the development process typically follow local policies with local security requirements, while the ramifications of a tool's or its user's action [6] (e.g. intentionally or un-intentionally committing corrupted code to a repository) can affect the entire project. Previous work like [3], further highlighted these specific challenges.

## 2.3 Methods for Security Requirements Elicitation

Frameworks like the Common Criteria (CC) [8], SSE-CMM [15] have been used successfully to elicit security requirements in a structured fashion. We chose SREP, an approach developed by Mellado et al. [10], for its integration of Common Criteria (CC) elements and a relatively light-weight execution process. For a more complete overview on requirements elicitation methods, we point to work by Tondel et al. [16].

## 2.4 Gaps in the Literature and Contribution

While there is an extensive research on security requirements engineering and elicitation processes, we see a lack of reported experiences on the practical application said research that would aid software project managers, developers and other software business practitioners in their operations. Furthermore, security conscious stakeholders in a GSE project lack categorization models for security requirements and reference requirements for GSE platforms.

We attempt to improve upon this situation by 1) reporting experiences from a real-world security requirements elicitation project on a GSE platform and 2) presenting the results in a structured manner that facilitates future work.

## 3 Systematic Elicitation of Security Requirements

The following sections reflect our experiences from applying Mellado's security requirements engineering process (SREP) [10] in the requirements elicitation project for the OpenCIT GSE platform. During the requirements elicitation project we used all of Mellado's 9 proposed steps, but placed special focuses on requirements-elicitation and categorization in steps 6 and 7. We further replace the security repository proposed by Mellado with a more light-weight approach based on misuse case descriptions. The process was conducted over the course of 2 months in Q4/2010 and involved 3 lead developers of the OpenCIT platform, 1 project manager, 2 security managers and 1 end-user.

### 3.1 Activity 1 - Agree on Security Definitions, Terminology

We adopt the definitions of vulnerability, threat, risk, exposure and countermeasure as outlined in the CISSP guide by Harris ([5], pp.58-59) due to their practitioner oriented focus and wide acceptance in the industry.

### 3.2 Activity 2 - Identify Vulnerable Assets

Based on the OpenCIT architecture (**Fig. 1**) we have identified three main categories of vulnerable and critical assets. The first one is the connector that enables communication and exchange of data with software engineering specific tools relevant in the engineering process. The second asset refers to EngSB-related components (e.g., workflow management) facilitating the execution engineering processes. The third main asset is the EngSB itself that enables message transfer between EngSB-related components and engineering tools.

### 3.3 Activity 3 - Identify Security Objectives and Dependencies

We elicit and prioritize security requirements for all three asset categories by adapting Fruehwirth et al.'s [2] approach for business-driven security objectives. The EngSB stakeholders first identified the security objectives in a brainstorming session, then weighted them through voting mechanism described in [2]. The results are compiled and listed in table 1.:

**Table 1.** Security objectives, assets and weighting

|  | Assets: | Tool Connectors | Bus | Component s |
|---|---|---|---|---|
| Objectives: | Weight(=I mportance) | 40% | 30% | 30% |
| Availability | 45% | HIGH | HIGH | MED |
| Integrity | 30% | MED | HIGH | HIGH |
| Confidentiality | 25% | LOW | MED | MED |

The table cells, express the targeted security level in respect to a particular security objective (e.g. "maintain high availability of tool connectors"). The concrete security levels associated with the terms HIGH, MED, LOW are defined by the service level agreements and security policy of the particular organization or its software engineering project on a case-by-case basis.

## 3.4 Activity 4 - Identify Threats and Develop Artifacts

Threats originate from threat agents that may harm a system through intentional or unintentional wrong-doing. Regular system users may become threat agents unintentionally by, for example, diverting from pre-defined engineering workflows. A common example is the case of software developers that put a system at risk by deploying changes of code or configurations, but forget to run dependency checks beforehand.

The concept of threat agents is well known in the literature [5] and can be used in combination with misuse cases to identify threats to a system. To identify threats relevant for a GSE platform like the OpenCIT, we 1) assume that a regular user may unintentionally become a threat agent at any point and 2) add a threat agent to all existing use cases descriptions of the OpenCIT documentation. We further created distinct misuse cases for threat agents that target a GSE system deliberately (as opposed to unintentionally), to describe their possible actions. This is in line with Mellado's [10] suggestion of developing misuse case artifacts during the threat analysis.

We compensate for the lack of knowledge about the threat agent's real attack motives with the assumption that his or her goal will at one point be to prevent the project from reaching its security objectives (elicited in Activity 3). Due to space constraints we exemplify the approach in Fig. 2 with the highest ranked misuse case: *"Compromise integrity of engineering artifact through forged messages"*.

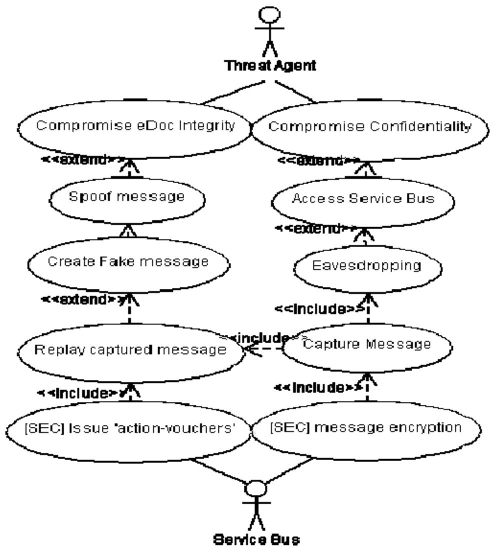

**Fig. 2.** An example misuse case, including threat activities, its adversary and countermeasures

In this use case (depicted in Fig. 2), the threat agent attempts a replay attack [14] to compromise the integrity of an engineering artifact (e.g. a source code file). To conduct a replay attack, the threat agent first eavesdrops on service bus transmissions until he has found and recorded a message that he intends to use in the attack. In this case, the threat agent captured a 'delete' command by a legitimate user. The threat agent can now attempt to spoof (mimic) a legitimate user request by re-using (i.e. replaying) either parts or the complete captured message.  In the "compromise integrity" use case, the threat agent would replay the captured 'delete' command, but change the URI of the resource to be deleted. In the absence of adequate security measures, a source code management system might accept the forged message, delete the indicated resources and thus compromise the integrity of the engineering artifact.

To prevent the threat agent from reaching his goal, the misuse case presents him with a second actor: the OpenCIT service bus. The OpenCIT service bus acts as an adversary, who deploys mitigation activities that the threat agent needs to overcome in order to fulfill his misuse case. The example in Fig. 2 shows two such mitigation activities or countermeasures. On the right side, the inclusion of "message encryption" in "capture message" forces the threat agent to first break the encryption of transmissions in order to complete his "capture message" activity. On the left, the inclusion of "Issue action vouchers" means that a disposable, 1-time-use ticket, or voucher, is attached to each message sent on the service bus. The 1-time-use voucher is verified by the bus and invalidated after a command has been successfully executed. If the threat agent replays a previously captured "delete" command, it contain an already used, thus invalidated, action voucher and would fail upon verification by the OpenCIT service bus.

A larger overview that integrates the misuse cases presented in Fig. 2 is available in the appendix of this paper and, along with the remaining cases, in [4].

We used developers' and domain experts' to develop misuse cases for the most likely paths a threat agent could take to achieve this goal (e.g. UC1: compromise code confidentiality by stealing user credentials. UC2: compromise code confidentiality by eavesdropping bus communication). Since various attack paths lead to the same goal, we organize them in a linked tree-model where each node represents an activity by the threat agent and a branch the complete attack path. The full misuse case tree is available in [4].

## 3.5 Activity 5 - Risk Assessment

Mellado [10] describes the purpose of the risk assessment with estimating "*the security risks based on the relevant threats, their likelihood and their potential negative impacts*". Using domain experts and developers for this task is likely to introduce personal bias in the outcome of the assessment. To at least partially compensate for personal bias, we briefed the risk assessors with security trend reports that we considered impartial, like Verizon's 2009 and 2010 data breach reports [13]. The purpose of the reports is to provide the risk assessors with empirical indicators on the likelihood and potential negative impacts of security incidents on a global scale, without specifically referring to the area of GSE. To our knowledge, similar empirical work has not yet been conducted in the context of GSE & security and should be considered for future research.

## 3.6 Activity 6 - Elicit Security Requirements

Activity 4 developed a set of misuse cases that were then categorized by the security objectives they threatened and prioritized by the level of perceived risk and organized in a tree model. Since every tree node represents an activity by the threat agent to reach his or her goal (i.e. the root node) we use the tree to identify the nodes that are critical for the misuse case to succeed. At these critical activities (e.g. "captured message"), we inserted possible mitigation activities or countermeasures (e.g. "encrypt message") that block the path and prevent the misuse case from succeeding.

We elicited the final security requirements by compiling a list of all inserted mitigation activities and describing their purpose in the misuse cases. The resulting security requirements are:

1) All message exchanges require the sender and recipient to authenticated themselves (2-way authentication)
2) All message exchanges need to be encrypted on the network level
3) All message exchanges need to be encrypted on the application / message content level
4) Every request for OpenCIT functionality (e.g. checkout or commit code) is subject to a credentials and access control check
5) Granted request for OpenCIT functionality shall be answered with a 1-time-use ticket, or 'action voucher' that can be redeemed by the owner at a 3rd party, in exchange for the 1-time execution of a task (e.g. delete file).
6) Local tools that are part of the software development process need to undergo integrity checks

Defensive programming guidelines need to be enforced in all parts of the OpenCIT system that accept application messages as input.

## 3.7 Activity 7 - Categorize Requirements

In contrast to the applied elicitation technique, we use a matrix of requirements defined by specific architectural levels and the given assets for structuring requirements, shown in Table 2, rather than a prioritization in terms of impact and likelihood of requirements only. The categorization of requirements takes into account several parts of the ISO/OSI model [17] down to the Transport Layer. Lower layers are meant to be future work for further investigation. In addition to the ISO/OSI model we further add three more categories. The *Workflows* category describes requirements related to engineering processes executed automatically and transparent to engineering tools. The *Tools* category explicitly takes care of engineering tools as they are closest to the engineer and as they play a significant part in an engineering process.

**Table 2.** Assignment matrix of requirements to assets at different levels of architecture

| | **Connector** | **EngSB Comp.** | **EngSB** |
|---|---|---|---|
| Tools | Tool Integrity Check | | |
| Workflows (Automated Processes) | AC-Service, 1-time voucher | AC-Service, 1-time voucher | AC-Service , Message encryption, 1-time voucher |
| Application Layer | AC-Service, 2-way auth., Defensive programming | AC-Service, 2-way auth., Defensive programming | AC-Service , Message encryption, 2-way auth., Defensive programming |
| Presentation Layer | | | |
| Session Layer | Traffic encryption | Traffic encryption | Traffic encryption |
| Transport Layer | Traffic encryption | Traffic encryption | Traffic encryption |

Based on the CI&T use case and the misuse cases described in [4] we identified techniques to be implemented in each cell of the matrix to counter those misuse cases. Traffic encryption between all communicating assets has to be enforced in order to prevent gaining access to OpenCIT by eavesdropping on network level. Message encryption has to be enforced in order to prevent eavesdropping at application level. Furthermore, tool integrity checks need to be implemented in order to make sure that engineering tools have not been manipulated. Access Control (AC) Services have to be deployed to enable checks regarding the credentials of requesters intending to execute specific actions. Two-way authentication is necessary in order to handle forged messages intending to take over someone's identity. A similar technique, one-time use action vouchers, prevents executing commands due to captured and then replayed communication messages.

## 3.8 Activity 8 - Requirements Inspection

The matrix developed in Activity 7 was used to perform a sanity check of the elicited requirements in an open workshop session with OpenCIT developers and the main stakeholders of a software engineering project had been used for at the time.

## 3.9 Activity 9 - Repository Improvement

Mellado [10] proposed to store security requirements in a Security Resource Repository (SRR). We chose to forgo the use of a full repository and made the requirements available online [4], in a more light weight and tool supported UML use case format, instead.

While we recognize an SRR as a practical storage format for security requirements, experiences with our industry partner showed that software developers and quality managers preferred a more visual, UML-type form for requirements storage. The main reason reported developers on why they preferred to get security requirements in the form of UML misuse cases was because its let them continue to use their familiar engineering tools. Software quality managers preferred UML for the ability to automatically deduct test cases, thus easing their quality & testing tasks.

Despite these experiences in the case of Open CIT, a growing number of requirements may require the use of SRR in the future, hence we plan to conduct further work in this area.

## 4 Experiences and Discussion

This section discusses the proposed approach and presents first gained impressions experienced by means of cooperation with an industry partner in developing a secure OpenCIT.

The applied process steps suggested by Mellado have been adapted in two different ways. Step 7 proposes a categorization and prioritization based on a qualitative ranking of each requirement whereas the requirement with the highest priority is handled first. In contrast to that, we believe that quantitative prioritization of requirements would be preferable with respect to a small team size in order to implement representative use cases much faster. In contrast to Mellano's suggestion to use an XML based Security Resource Repository (SRR) in step 9, we applied UML. In that particular industry context UML has been favored over SRR because SRR was too heavy weight and too complex for a project of our size. Additionally, OpenCIT developers lacked specialized security skills but were already experts and daily users of UML style use cases, user stories, or requirements elicitation. This approach proved to be complete and practically applicable even for non-security experts. Nevertheless of UML, the proposed approach still facilitates accountability regarding implementations of security requirements, and thus demonstrates an impact on software quality this way. Furthermore, it helps identify security aspects which are still missing but required in the design of the architecture and consequently in the implementation.

From our experience in developing a secure implementation of OpenCIT, we believe that it is essential to cooperate with domain experts (i.e. in our case with an industry partner) for a successful creation of misuse cases as they are likely able to put themselves in threat agent's shoes. With respect to our approach, we have understood that it works well when key requirements are identified, but lacks strength when niche- or minor requirements (e.g., length of a specific encryption key to be used for en/decryption) are discussed. However, those requirements may be reflected in a tree-like structure and put as additional description to a specific set of key requirements. Additionally, as noted by the industry partner, further work of the alignment matrix should include aligning security requirements with business goals (i.e. economic considerations) to track requirements to code more effectively and efficiently.

## 5 Conclusion and Future Work

For the successful and efficient realization of global software engineering projects the interconnection of distributed, multi-engineering environments and their tools is essential. Furthermore, such projects need to rely on a sufficient level of security to be sustainable and capable of to managing technical, legal and business risks that arise from distributed development. However, the definition of a level is typically captured in implicit requirements and rarely gathered in a methodological way resulting in an unclear/undefined security state regarding operations implemented by developers.
In this paper we reported on our experiences in applying standard requirement elicitation techniques regarding security aspects. The intention was to elicit requirements in methodological manner enabling a structured storage, retrieval, or checking of requirements. We used the OpenCIT platform, a continuous integration and testing framework widely used in software engineering projects, to present our report. Based on a list of gathered security requirements for secure collaboration of software engineers in global software development environments, we identified key security objectives and dependencies to gain a list of prioritized requirements (e.g., availability is more important than confidentiality). The combination of architectural assets worth protecting and a layering of the architecture enabled to find and define specific techniques of protecting the framework. It seems that although the number of gathered security requirements is low, they still have to be coped with at each defined asset of the architecture. Nevertheless, those specific requirements already cover a relatively large number of misuse cases. However, as the OpenCIT platform is based on a message system, we only investigated security issues related to communication and data exchange.

Future work will include investigation and evaluation of the proposed assignment matrix in large software engineering projects to verify its usability, in multi-vendor engineering projects using software product lines (SPL) to identify its advantages and limitations, and in multi-engineering environments. Furthermore, we will analyze the impact on the presented approach regarding the complexity of UML descriptions, and thus we will investigate its scalability.

**Acknowledgments.** This work has been supported by the Christian Doppler Forschungsgesellschaft, the BMWFJ, Austria and TEKES, the Finnish Funding Agency for Technology and Innovation.

## References

1. Biffl, S., Mordinyi, R., Moser, T.: Automated Derivation of Configurations for the Integration of Software(+) Engineering Environments. Paper presented at the 1st International Workshop on Automated Configuration and Tailoring of Applications, ACoTA 2010 (2010)
2. Fruehwirth, C., Biffl, S., Tabatabai, M., Weippl, E.: Addressing misalignment between information security metrics and business-driven security objectives. Paper presented at the Proceedings of the 6th International Workshop on Security Measurements and Metrics, Bolzano, Italy (2010)

3. Frühwirth, C., Biffl, S., Schatten, A., Schrittwieser, S., Weippl, E., Sunindyo, W.: Research Challenges in the Security Design and Evaluation of an Engineering Service Bus Platform. Paper presented at the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Lille, France (2010)
4. Frühwirth, C., Mordinyi, R., Biffl, S.: Systematic Definition of Security Requirements by means of Misuse Cases in Multi-Engineering Domains, Christian Doppler Laboratory, Vienna University of Technology (2011), `http://cdl.ifs.tuwien.ac.at/techrep/icgse`
5. Harris, S.: CISSP All-in-One Exam Guide. McGraw-Hill (2008)
6. Herbsleb, J.D.: Global Software Engineering: The Future of Socio-technical Coordination. Paper presented at the 2007 Future of Software Engineering (2007)
7. Kang, M.H., Park, J.S., Froscher, J.N.: Access control mechanisms for inter-organizational workflow. Paper presented at the Proceedings of the sixth ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, United States (2001)
8. Keblawi, F., Sullivan, D.: Applying the Common Criteria in Systems Engineering. IEEE Security and Privacy 4(2), 50–55 (2006), doi:10.1109/msp.2006.35
9. Long, D.L., Baker, J., Fung, F.: A prototype secure workflow server. In: Proceedings of 15th Annual Computer Security Applications Conference (ACSAC 1999), pp. 129–133 (1999)
10. Mellado, D., Fern, E., Medina, N., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput Stand Interfaces 29(2), 244–253 (2007), doi:10.1016/j.csi.2006.04.002
11. Mordinyi, R., Moser, T., Biffl, S., Dhungana, D.: Flexible Support for Adaptable Software and Systems Engineering Processes. Paper presented at the Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), USA (2011)
12. Moser, T., Biffl, S.: Semantic Tool Interoperability for Engineering Manufacturing Systems. Paper presented at the 15th IEEE International Conference on Emerging Techonologies and Factory Automation (ETFA 2010) (2010)
13. Moser, T., Mordinyi, R., Sunindyo, W.D., Biffl, S.: Semantic Service Matchmaking in the ATM Domain Considering Infrastructure Capability Constraints. In: Du, W., Ensan, F. (eds.) Canadian Semantic Web: Technologies and Applications, pp. 133–157. Springer, Heidelberg (2010)
14. Mut-Puigserver, M., Payeras-Capellà, M.M., Ferrer-Gomila, J.L., Huguet-Rotger, L.: Replay Attack in a Fair Exchange Protocol. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 174–187. Springer, Heidelberg (2008)
15. Systems Security Engineers - Capability Maturity Model, `http://www.sse-cmm.org/index.html`
16. Tondel, I.A., Jaatun, M.G., Meland, P.H.: Security Requirements for the Rest of Us: A Survey. IEEE Softw. 25(1), 20–27 (2008), doi:10.1109/ms.2008.19
17. Zimmermann, H.: OSI reference model\&mdash;The ISO model of architecture for open systems interconnection. In: Innovations in Internetworking, pp. 2–9. Artech House, Inc. (1988)

## Appendix

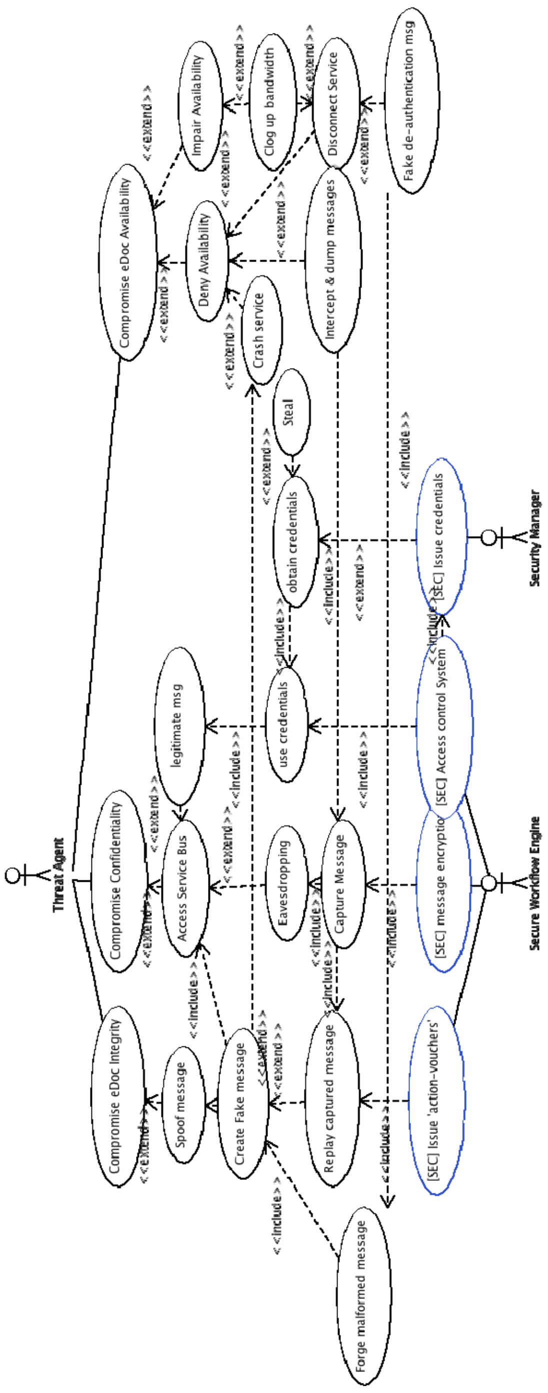

**Fig. 3.** Misuse Case Overview

# Author Index