Research Reports in Software Engineering and Management 2007:02

Proceedings of the 7th Conference on Software Engineering Research and Practice in Sweden

Thomas Arts (Ed.)

IT University of Göteborg
CHALMERS | GÖTEBORGS UNIVERSITET

Department of Applied IT

Proceedings

Seventh Conference on Software Engineering Research and Practice in Sweden

SERPS'07

24-25 October, 2007
IT University of Göteborg, Göteborg, Sweden

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY and CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2007
ISSN: 1654-4870

Research reports in Software Engineering and Management
Report number 2007:02

Series editor: Lars Pareto

www.ituniv.se/sem_research

Preface

Welcome to SERPS'07 – the seventh conference on Software Engineering Research and Practice in Sweden.

This year we selected 11 papers and 2 thesis abstracts of in total 15 submissions. We got submissions in all six main topics, viz Requirements Engineering, Product and Project Management, Design, Quality Management, Verification and Validation, and Methods. This indicates that all research is performed over the whole scale of topics that SERPS covers.

As in other years, we obtained a good number of papers with direct industrial participation. This shows that the conference can accommodate papers that have one foot in research and one in practice, which we are very happy for.

SERPS offers a forum to young researchers to submit their papers and get valuable feedback for improving before they send it to another workshop, conference, or journal. Many of these contributions have nowadays a high standard, already when submitted to SERPS. The conference also provides the opportunity to listen to research presentation of other researchers in the field and as such create a strong national awareness of research going on in Sweden. For that reason SERPS welcomes papers from more senior researchers, such that the conference is representative for the research performed in Sweden. That in its turn allows us to market the conference towards industry as: "come and see what is happening in Software Engineering research in Sweden".
I like to thank all authors for their submissions, vital for the existence of a conference.

Reviewing papers for SERPS is not so much selecting the good papers as well as providing valuable feedback to the authors of the papers to make their submission into a good paper. I would like to thank all reviewers for their important contribution.

I would also like to express my appreciation to Miroslaw Staron for the local organization of the conference. Coffee, lunches, a conference dinner, all those issues make a conference a pleasant event if it works smoothly and a bit cumbersome if things do not work out. Thanks Mirek for taking care of it! Thanks also to Linda Kullenberg for creating the website.

Have fun at the conference

Thomas Arts
Program chair

Conference Organization

General chair
Peter Öhman, IT University of Göteborg

Program chair
Thomas Arts, IT University of Göteborg

Program Committee
Jonas Andersson, Syntell AB
Thomas Arts, IT University of Göteborg
Tomas Berling, Saab Microwaves
Jürgen Börstler, Umeå University
Ivica Crnkovic, Mälardalen University
Robert Feldt, Blekinge Institute of Technology
Martin Höst, Lund Institute of Technology
Mira Kajko-Mattsson, Stockholm University
Michael Mattsson, Blekinge Institute of Technology
Peter Öhman, IT University of Göteborg
Anne Persson, Skövde University
Per Runeson, Lund Institute of Technology
Kristian Sandahl, Linköping University
Martin Törngren, KTH - Royal Institute of Technology
Claes Wohlin, Blekinge Institute of Technology

Additional reviewers
Aneta Vulgarakis, Mälardalen University
Séverine Sentilles, Mälardalen University
Hongyu Pei-Breinvold, Mälardalen University
Magnus Persson, KTH - Royal Institute of Technology

Local arrangements
Miroslaw Staron

Website
Linda Kullenberg

Table of Contents

Thesis Abstract: Key Elements of Software Product Integration Processes

Stig Larsson
Mälarsdalen University
Västerås, Sweden
stig.larsson@mdh.se

ABSTRACT

The integration phase represents a highly critical part of the software product development process as components are combined and should work together. Errors and problems in product integration result in delays and rework as the resulting artifacts are needed for later phases. Standards and other reference models that include guidelines for product integration are available, but are not always used.

Our proposal is that is that the current descriptions in standards and reference models are taken one by one insufficient and need to be consolidated to help development organizations improve the product integration process. The presented research includes a number of case studies and analyses that have resulted in a union of product integration practices, i.e. a combination of the activities included in the different reference models. Through the case studies performed in seven different product development organizations, a connection between problems that are observed and the failure to follow the recommendations is identified. The analysis has indicated which practices are necessary, and how other practices support these. We have also found a connection between the development of software architectures and how that product integration practices need to be adapted when evolving products and systems, and provide organizations with a method to find necessary adaptations

1. MOTIVATION

Good practices for product integration are described and made available through different reference models and standards such as ISO/IEC 12207 [1], CMMI [2], EIA-731.1 [3], and ISO/IEC 15288 [4]. Results from research investigating costs related to different phases [5], integration in relation to testing [6], and in why available methods are underused [7] as well as my own experience suggest that the available knowledge is not always utilized, or that the recommendations in the reference models are insufficient. This leads to inadequate, insufficient, or lacking use of activities that would ensure efficient and effective product integration. are not used may be that they sometimes are not fully understood or that they are perceived as not being applicable for specific organization, development models used.

Failure in the integration can thus be expensive and need to be avoided. Practices described in different reference models may help in avoiding these problems and can be divided into three categories:

- Preparation of product integration
  This includes decisions on strategy, integration sequence and of the criteria for integration
- Management of interfaces between components
  The integration processes include checking that interfaces are properly defined, and that changes to interfaces are controlled, but not the definition and design of the interfaces as this is a design issue
- Execution of the product integration
  The execution comprise ensuring that the strategy, sequence and criteria are followed, assembling the components, as well as performing planned tests

However, the specifics of the reference models differ, and there is a need to understand how these differences may affect the performance of product development projects. Another important aspect is to understand what is needed to help organizations to better follow reference models in different product integration undertakings.

This leads to the objectives for this research: to find *what practices among those available in reference models help product development units avoiding problems in product integration and making it efficient and effective*. We would also like to understand if the reference models are sufficient or if there are other means to help organizations to improve the execution of product integration.

2. RESEARCH APPROACH AND RESULTS

To investigate the factors that influence the software product integration processes, we have used different types of reference models. We have examined what effect the use of, or negligence of following, the practices described in the reference models have on the performance in product integration. This is done in investigations of product development organizations through examining development projects. In addition to this, we have examined how changes in architecture can influence processes, and how this influence can be captured.

Through a combination of an analysis of the reference models, and a compilation of seven different industrial cases, we have identified 15 practices that are useful for efficient and effective product integration. The cases are described in [8-10]. A compilation of the results are presented in detail in [11], and are summarized here. The reference model analysis resulted in a union consisting of 15 practices which describes what can be considered the current level of knowledge in product integration.

Of the 15 practices four are concerned with preparation of the product integration:

1. Define and document an integration strategy
2. Develop a product integration plan based on the strategy
3. Define and establish an environment for integration
4. Define criteria for delivery of components

The following five practices describe design and interface management:

5. Identify constraints from the integration strategy on design
6. Define interfaces
7. Review interface descriptions for completeness
8. Ensure coordination of interface changes
9. Review adherence to defined interfaces

One practice defines the preparation of the verification to be performed in the product integration:

10. Develop and document a set of tests for each requirement of the assembled components

The actual integration of components is made up of four practices:

11. Verify completeness of components obtained for integration through checking criteria for delivery
12. Deliver/obtain components as agreed in the schedule
13. Integrate/assemble components as planned
14. Evaluate/test the assembled components

Finally, a single practice ensures that the integration is documented:

15. Record the integration information in an appropriate repository

Of these Product Integration practices, we have observed that problems are likely if any of the following five are neglected: PI practices 4, 7, 8, 11, and 12. However, the practices are not independent and the set of practices that need to be followed is larger than the set that we have seen causes problems in the development organizations as they support the crucial ones.

When investigation the product integration area, we have seen that organizations are aware of practices that are described in reference models. However, as the information in the models is too limited, the usefulness is limited and additional information such as examples and hands-on methods are needed. Consequently, the models should primarily be used as guidelines for what to improve, and information about how the practices should be implemented need to be found elsewhere.

One observation was made in the case studies: the architecture of a product or system is very often changed, but the processes to further develop the system are not altered to reflect this evolvement. Through an investigation of different models used for supporting architectural decisions, and appraisal methods for process improvement, a method has been proposed and piloted [12]. The method was successful in helping the organization to understand what process changes are needed to benefit from the architectural changes. This was especially true for the product integration process as the architectural changes called for new strategies.

3. CONCLUSIONS

Product integration enables an organization or a project to observe all important attributes that a product will have; functionality, quality and performance. This is especially true for software systems as the integration is the first occurrence where the full result of the product development effort can be observed. Consequently, the integration phase represents a highly critical part of the product development process. Although reference models that describes practices for product integration, research and experiences indicate that practices are not used in an effective manner.

Through case studies covering seven different product development organizations, the ineffective use of practices that are described in reference models have been connected to the problems that have been observed. Our analysis indicates that the management of interfaces as well as the delivery of components that fulfill criteria are crucial to the effectiveness and efficiency of software product integration. We have also found a connection between the development of software architectures and how that product integration practices need to be adapted when evolving products and systems, and have proposed and piloted a method to find necessary adaptations.

Additional research is needed to look at other methods, tools, and technologies to help product development organizations improve product integration. Based on the available reference models and understand how these can help, a foundation is available for future research. Also, through providing a method to understand how different changes affect the processes, proposed improvements in the means for better product integration can be understood and assessed.

4. REFERENCES

[1] ISO/IEC12207:1995, "Information technology - Software life cycle processes," ISO/IEC, 1995.

[2] SEI, "CMMI® for Development, Version 1.2.," Pittsburgh, PA, USA, Technical Report CMU/SEI-2006-TR-008, 2006.

[3] EIA-731.1, "Systems Engineering Capability Model," Electronic Industries Alliance, 2002.

[4] ISO/IEC15288:2002, "Systems engineering - Systems life cycle processes," ISO/IEC, 2002.

[5] J. Campanella, Principles of Quality Costs: Principles, implementation and Use, 3rd ed. Milwaukee, WN, USA,: ASQ Press, 1999.

[6] RTI, "The Economic Impacts of Inadequate Infrastructure for Software Testing." Gaithersburg, MD, USA,: National Institute of Standards and Technology, 2002.

[7] M. Bajec, D. Vavpoti, and M. Krisper, "Practice-driven approach for creating project-specific software development methods," Information and Software Technology, vol. 49, pp. 345, 2007.

[8] S. Larsson, I. Crnkovic, and F. Ekdahl, "On the expected synergies between component-based software engineering and best practices in product integration," presented at Proceedings - 30th EUROMICRO Conference, Aug 31-Sep 3 2004, Rennes, France, 2004.

[9] S. Larsson and I. Crnkovic, "Case Study: Software Product Integration Practices," presented at 6th international conference Profes, June, 2005, Oulu Finland, 2005.

[10] S. Larsson, P. Myllyperkiö, and F. Ekdahl, "Product Integration Improvement Based on Analysis of Build Statistics," presented at ESEC/FSE, Dubrovnik, Croatia, 2007.

[11] S. Larsson, P. Myllyperkiö, F. Ekdahl, and I. Crnkovic, "Examination of Product Integration Practices in Reference Models," Information & Software Technology, 2007.

[12] S. Larsson, A. Wall, and P. Wallin, "Assessing the Influence on Processes when Evolving the Software Architecture," presented at IWPSE 2007, Dubrovnik, Croatia, 2007.

[13]

A Classification Framework for Component Models

Ivica Crnkovic
Mälardalen University, Department of Computer Science and Electronics
Box 883, SE-721 23 Västerås, Sweden
ivica.crnkovic@mdh.se

Michel Chaudron
Technical University Eindhoven, Dept. of Mathematics and Computing Science,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
m.r.v.chaudron@TUE.nl

Séverine Sentilles
Mälardalen University, Department of Computer Science and Electronics
Box 883, SE-721 23 Västerås, Sweden
severine.sentilles@mdh.se

Aneta Vulgarakis
Mälardalen University, Department of Computer Science and Electronics
Box 883, SE-721 23 Västerås, Sweden
aneta.vulgarakis@mdh.se

ABSTRACT
The essence of component-based software engineering is embodied in component models. Component models specify the properties of components and the mechanism of component compositions. In a rapid growth, a plethora of different component models has been developed, using different technologies, having different aims, and using different principles. This has resulted in a number of models and technologies which have some similarities, but also principal differences, and in many cases unclear concepts. Component-based development has not succeeded in providing standard principles, as for example object-oriented development. In order to increase the understanding of the concepts, and to easier differentiate component models, this paper provides a Component Model Classification Framework which identifies and quantifies basic principles of component models. Further, to illustrate its utilization, this paper also classifies a certain number of component models using this framework.

Categories and Subject Descriptors
D.2.2 Design Tools and Techniques

General Terms
Design, component-based software engineering.

Keywords
Component models, taxonomy.

1. INTRODUCTION
Component-based software engineering (CBSE) is an established area of software engineering. The inspiration for "building systems from components" in CBSE comes from other engineering disciplines, such as mechanical or electrical engineering, and Software Architecture in which a system is seen as a structure with clearly identified components and connectors. The techniques and technologies that form the basis for component models originate mostly from object-oriented programming and Architecture Description Languages (ADLs). Since software is in its nature different from the physical world, the translation of principles from the classical engineering disciplines into software is not trivial. For example, the understanding of the term "component" has never been a problem in the classical engineering disciplines, since a component can be intuitively understood and that understanding fits well with fundamental theories and technologies. This is not the case with software; the notation of a software component is not clear: its intuitive perception may be quite different from its model and its implementation. From the beginning, CBSE struggled with a problem to obtain a common and a sufficiently precise definition of a software component. An early and probably the most commonly used definition coming from Szyperski [1] ("*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party*") focuses on characterization of a software component. In spite of its generally it was shown that this definition is not valid for a wide range of component-based technologies (for example those which do not support contractually specified interface, or independent deployment). In the definition of Heineman and Councill [2] ("*A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard*"), the component definition is more general – actually a component is specified through the specification of the component model but the component model itself is not specified. This definition of a component can be even more pushed further in the generalization, but on the contrary the definition of a component model can be expressed more precisely [3]:

Definition I: A *Software Component* is a software building block that conforms to a component model.

Definition II: A *Component Model* defines standards for (i) properties that individual components must satisfy and (ii) methods, and possibly mechanisms, for composing components.

This generic definition allows the existence of a wide spectrum of component models, which is also happening in reality; there exist many component models with many different characteristics on the market and in different research communities. This diversity makes it more difficult to properly understand the Component-Based (CB) principles, and to properly select a component model of interest, or to compare models. In particular, this is true since CB principles are not clearly explained and formally defined. In their diversities component models are similar to ADLs; there are similar mechanisms and principles but very different implementations. For this reason there is a need for providing a framework which can provide a classification and comparison between different component models in a similar manner as it was done for ADLs [4,5].

In this paper, we thus propose a classification and comparison framework for component models. Since component models and their implementations in component technologies cover a large range of different aspects of the development process, we group

these aspects in several dimensions of the framework - for certain component models we will say that they are similar in one dimension, but different in another. Several different taxonomies of component models already exist. An example is [6] in which taxonomy is described in respect to compositions and component life cycle. Another example is [7] in which the emphasis is on reuse aspects and characteristics of different application domains. Our comparison framework has the goal to provide a multidimensional framework, that counts different, yet equality important aspects of component models.

The remainder of this paper is as follows. Section two motivates, explains and defines the different dimensions of the classification framework. Section three gives a very brief overview of selected component models, and section four provides a short description of component model characteristics in the comparison framework, for each dimension.

2. The Classification Framework

The main concern of a component model is to (i) provide the rules for the specification of component properties and (ii) provide the rules and mechanisms for the component composition, including the composition rules of component properties. These main principles hide many complex mechanisms and models, and have significant differences in approaches, concerns and implementations. For this reason we cannot simply list all possible characteristics to compare the component models; rather we want to group particular characteristics that have similar concerns i.e. that describe the same or related aspects of component models. The fundamental principles can be divided into the following categories:

1. **Lifecycle.** *The lifecycle dimension identifies the support provided (explicitly or implicitly) by the component model, in certain points of a lifecycle of components or component-based systems.* Component-Based Development (CBD) is characterized by the separation of the development processes of individual components from the process of system development. There are some synchronization points in which a component is integrated into a system, i.e. in which the component is being bound. Beyond that point, the notion of components in the system may disappear, or components can still be recognized as parts of the system.

2. **Constructs.** *The constructs dimension identifies (i) the component interface used for the interaction with other components and external environment, and (ii) the means of component binding and communication.* Interface specification is the characteristic "sine qua non" of a component model. In some component models, the interface comprises the specification of all component properties, but in most cases, it only includes a specification of properties through which the communication with the environment should be realized. Directly correlated to the interface are the components' interoperability mechanisms. All these concepts are parts of the "construction" dimension of CBD.

3. **Extra-Functional Properties.** *The extra-functional properties dimension identifies specifications and support that includes the provision of property values and means for their composition.* In certain domains (for example real-time embedded systems), the ability to model and verify particular properties is equally important but more challenging than the implementation of functional properties themselves.

4. **Domains.** *This dimension shows in which application and business domains component models are used.* It indicates the specialisation, or the generality of component models.

In these four dimensions, we comprise the main characteristics of component models but, of course, there are other characteristics that can differentiate them. For example, since in many cases component models are built on a particular implementation technology, many characteristics come directly from this supporting implementation technology and that are not visible in component models themselves.

2.1 Lifecycle

In the software development lifecycle, a number of methods and technologies specifying and supporting particular phases of the cycle exist. While CBSE aims at covering the entire lifecycle of component-based systems, component models provide only partial lifecycle support and are usually related to design, implementation and integration phases.

The overall component-based lifecycle is separated into several processes; building components, building systems from components, and assessing components [6]. Some component technologies provide certain support in these processes (for example maintaining component repositories, exposing interface, and similar).

The component-based paradigm (i.e. composability and reusability) has extended the integration activities in the run-time phase; certain component technologies provide extended support for dynamic and independent deployment of components into the systems. This support reflects the design of many component models. Accordingly, some of the components are only available at development stage and at run-time the system is monolithic. However not all component models consider the integration phase. We can clearly distinguish different component models that are related to a particular phase and such phase can be different for different component models. Some component technologies start in the design stage (e.g. Koala which has an explicit and dedicated design notation). Many other component technologies focus on the implementation phase (e.g. COM, EJB). For this reason one important dimension of the component model classification is the lifecycle support dimension. In such classification, we must consider both component lifecycle and component-based system lifecycle, which are somewhat different [3, 9] and are not necessary temporally related – they are ongoing in parallel and have some synchronization points. Here we identify characteristic "points" of both lifecycles that are concerns in component models:

(i) Modelling stage. The component models provide support for the modelling and the design of component-based systems and components. Models are used either for the architectural description of the systems and components (e.g. ADLs), or for the specification and the verification of particular system and component properties (e.g. statecharts).

(ii) Implementation stage. The component model provides support for generating and maintaining code. The implementation can stop with the provision of the source code, or can continue up to

the generation of a binary (executable) code. The existence of executable code is an assumption for the dynamic deployment of the components (i.e. the deployment of the components in the system run-time).

(iii) Packaging stage. Since components can be developed separately from systems, and the primary idea of the component-based approach is to reuse existing components, there is a need for their storage and packaging – either in a repository or for distribution. The result of this stage can be a file, archive, or a repository where the packaged components, including documentation and specification, are residing prior to decisions about how they will be run in the target environment. For example, in Koala, components are packed into a file system-based repository, in which a directory exists for each component. The directory contains a Component Description Language (CDL) file and a set of C and header files. Additionally, it can also contain interface definition files and/or data definition files. Another example of packaging is achieved in the EJB component model. There, packaging is done through jar archives, called ejb-jar. Each archive contains an XML deployment descriptor, a component description, a component implementation and interfaces.

(iv) Deployment stage. At a certain point of time, a component is integrated into a system i.e. bound to the execution platform. This activity happens at different points of development or maintenance phase. However, each of the component technologies that exist today solves the deployment issues in their own particular way. In general, the components can be deployed at compilation time (static binding) as part of the system, making it no longer possible to change how the components interact with each other, or at run time as separate units by using means such as registers (COM) or containers (CCM, EJB). For instance, Koala components are deployed at compilation time and they use static binding by following naming conventions and generated renaming macros. In opposition, CORBA components are deployed at run time in a container by using the information of the deployment descriptor packed with the component implementation.

2.2 Constructs

As mentioned in [30], the verb "construct" means "to form something by putting different things together", so in applying this definition to the CBSE domain, we define under this "Constructs" dimension, the way components are connected together within a component model in order to provide communication means. But although this communication aspect is of primordial importance, it is not often expressed explicitly. Instead, it is reflected implicitly by some underlying mechanisms. This is at contrary to functional – and sometimes extra-functional – properties of a component which are clearly stated in component interfaces. Consequently, a component interface has a double role: it first specifies the component properties (functional and possibly extra-functional), and second, it defines the actions through which components may be interconnected. Some of the component models distinguish also the "provides"-part (i.e. the specification of the functions that the component offer) from "requires"-part (i.e. the specification of the functions the component require) of an interface.

Besides coming along with the massive emergence of component models, several languages exist nowadays for specifying an interface: modelling languages (such as UML or different ADLs), particular specification languages (Interface Definition Languages), programming languages (such as interfaces in Java) or some additions built directly in a programming language. Similarly, the interaction can also be of different types: port-based where ports are the channels for communication of different data types and events; functions in programming languages defining input and output parameters; interfaces or classes in Object Oriented programming languages.

However, an interface remains most of time a very succinct description of the services a component proposes or needs. So in order to ensure that a component will behave as expected according to its specification and operational mode, the notion of contract has been adjoined to interfaces. According to [10], contracts can be classified hierarchically in four levels which, if taken together, may form a global contract. We only adopt the three first levels in our classification since the last level "contractualizes" only the extra-functional properties and this is not in direct relation with interoperability

- *Syntactic level:* describes the syntactic aspect, also called signature, of an interface. This level ensures the correct utilisation of a component. That is to say that the "client-component" must refer to the proper types, fields, methods, signals, ports and handles the exceptions raised by the "server-component". This is the most common and most easy agreement to certify as it relies mainly on an, either static or dynamic, type checking technique.
- *Semantic level:* reinforces the previous level of contracts in certifying that the values of the parameters as well as the persistent state variables are within proper ranges. This can be asserted by pre-conditions, post-conditions and invariants. A generalization of this level can be assumed as semantics.
- *Behaviour level:* dynamic behaviour of services. It expresses either the composition constraints (e.g., constraints on their temporal ordering) or the internal behaviour (e.g. dynamics of internal states).

Finally, the constructs dimension refers to the notions of reusability and evolvability, which are important principles of CBSE. Indeed, many component models are endowed with diverse features for supporting them but one typical solution is directly related to the existence of interfaces and therefore to our constructs dimension. This solution offers the ability to add new interfaces to a component which makes possible to embody several versions or variants of functions in the component.

Another type of binding is also realised through connectors. Connectors, introduced as distinct elements in ADLs, are not common among the first class citizens in most component models. Connectors are mediators in the connections between components and have a double purpose: (i) enabling indirect composition (so-called exogenous composition, in regards to direct or endogenous composition), (ii) introducing additional functionality. Exogenous composition enables more seamless evolution since it allows independent changes of components. In addition, in several component technologies, connectors act as specialised components, such as adaptors or proxies, either to provide

additional functional or extra-functional properties, or to extend the means of intercommunication.

The interface specification implicitly defines the type of interaction between components to comply with particular architectural styles. In most cases, particular component models provide a single basic interaction mechanism, but others, such as Fractal for example, allow the construction of different architectural styles.

For the constructs dimension of this classification framework, we distinguish consequently the following points.

(i) *Interface specification,* in which different characteristics allowing the specification of interfaces are identified:
   a. The distinction between the notions of interface and port. Although a port is generally seen as a part of an interface, in some component models a port is actually the only mean of communication. In these cases, the binding is done in a wiring manner such as in the pipe and filter architectural style. On the contrary, interfaces may involve different ways of binding, for example function calls, or queries.
   b. The distinction between the provides-part and requires-part of an interface.
   c. The existence of some distinctive features appearing only in this component model; and,
   d. The language used to specify those interfaces.

(ii) *Interface levels* which describe the levels of contractualisation of the interfaces, namely syntactic, semantic and/or behaviour level

(iii) *Architectural Style* which aims at identifying the recurrent patterns of interaction among components. Some of them are for example pipe&filter or client/server.

(iv) *Communication type* which details mainly if the communication used is synchronous and/or asynchronous. An extension of this could be to consider also the number of receivers (unicast, multicast or broadcast).

(v) *Binding type* describes the way components may be linked together through the interfaces.
   a. The exogenous sub-category depicts if the component model allows using some connectors. and,
   b. The vertical sub-category expressing the possibility of having a hierarchical composition of components

We assume here that the "endogenous" composition and the "horizontal" binding are the default mechanism of any component model, i.e. a "direct" connection between two components.

2.3 Extra-Functional Properties

Properties (also designated as attributes) are used in the most general sense as defined by standard dictionaries, e.g.: "a construct whereby objects and individuals can be distinguished" [11]. There is no unique taxonomy of properties, and consequently there can exist many property classification frameworks. One commonly used classification is to distinguish functional from extra-functional properties. While functional properties describe functions or services of an object (individual or thing), extra-functional properties (EFP) specify the quality (in a broader sense) of objects. In CBSE, there is a distinction between component properties and system properties. The system properties can be the result of the composition of the same properties of components, but also of a composition of different properties [12]. Important concerns of CBSE are how to provide relevant parameters from components which will be used in a provision of the system properties.

The two main dimensions in which component models differ in the way they manage EFP are the following:

– A property is managed by the system (exogenous EFP management) or managed by components (endogenous EFP management). This corresponds to wonder which actor manages a property;

– A property is managed on a system-wide scale or the property is managed on a per-collaboration basis (i.e. what is the scope of management of a property).

The different types of approaches are characterized by the reference architectures shown in Figure 1

Many component models provide no specific facilities for managing extra-functional properties. The way a property is handled is left to the designers of the system, and as a result a property may not be managed at all (approach A). This approach makes it possible to include EFP management policies that are optimized towards a specific system, and also can cater for adopting multiple policies in one system. This heterogeneity may be particularly useful when COTS components need to be integrated. On the other hand, the fact that such policies are not standardized may be a source of architectural mismatch between components.

The compatibility of components can be improved if the component model provides standardized facilities for managing

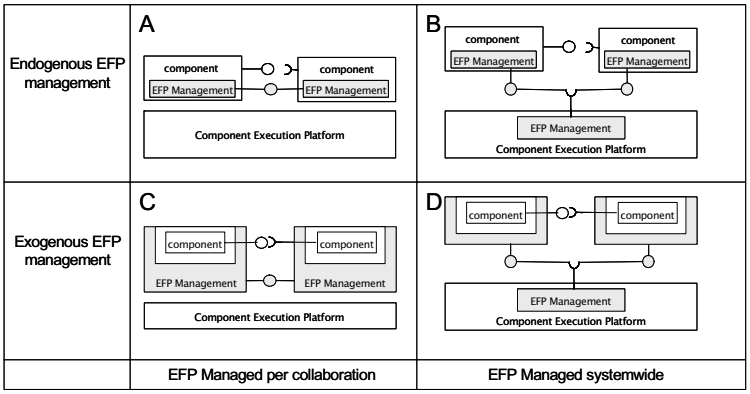


Figure 1. Management of extra-functional properties

EFP (approach B in Figure 1). In this approach, there is a mechanism in the component execution platform that contains policies for managing EFP for individual components as well as for EFP involving multiple components. The ability to negotiate the manner in which EFP are handled requires that the components themselves have some knowledge about how the EFP affects their functioning. This is a form of reflection.

A third approach is that the components should be designed such that they address only functional aspects and not EFP. Consequently, in the execution environment, these components are surrounded by a container. This container contains the knowledge on how to manage EFP. Containers can either be connected to containers of other components (approach C) or containers can interact with a mechanism in the component execution platform that manages EFP on a system wide scale (D).

The container approach is a way of realizing separation of concerns in which components concentrate on functional aspects and containers concentrate on extra-functional aspects. In this way, components become more generic because no modification is required to integrate them into systems that may employ different policies for EFP. Since these components do not address EFP, another advantage is that they are simpler and smaller and hence they are cheaper to implement.

For the EFP we provide a classification in respect to the following questions:

(i) *Extra-functional properties support*: does the component model provide general principles, means and/or support for specification and reasoning about extra-functional properties?

(ii) *Extra-functional properties specification*: Does the component model contain means for specification and reasoning of specific extra-functional properties. If yes, which types and/or which properties?

(iii) *Composability of extra-functional properties*: Does the component model provide means, methods and/or techniques for composition of certain extra-functional properties. If yes, which properties and/or what type of composition?

2.4 Domains

Some component models are aimed at specific application domains as for instance consumer electronics or automotive. In such cases, requirements from the application domain penetrate into the component model. As a result, the component model provides a natural fit for systems in that particular domain. The benefits of a domain-specific component models are that the component technology facilitates achieving certain requirements. Such component models are, as a consequence, limited in generality and will not be so easily usable in domains that are subject to different requirements.

Some component models are of general-purpose. They provide basic mechanisms for the production and the composition of components, but on the other hand, provide no guidance, nor support for any specific architecture. A general solution that enables component models to be both generally applicable but also cater for specific domains is through the use of optional frameworks. A framework is an extension of a component model that may be used, but is not mandatory in general.

There is a third type of component models - namely generative; they are used for instantiation of particular component models. They provide common principles, and some common parts of technologies (for example modelling), while other parts are specific (for example different implementations).

3. SURVEY OF COMPONENT MODELS

Nowadays there are numerous component models which can vary widely in many possible aspects: In usage, in support provided, in concerns, in complexity, in formal definitions and similar. In our classification of component models, the first question is whether a model (or technology, method, or similar) is a component model or not. Similar to biology in which viruses cover the border between life and non-life, there is a wide range of models, from those having many elements of component models but are still not assumed as component models, via those that lack many elements of component models, but are still called component models, through to those which are assumed as being component models. Therefore, we identify the minimum criteria required to classify a model, or a notation as a component model. This minimum is defined by Definition I and Definition II: A model that explicitly or implicitly identifies components and defines rules for specification of component properties and means of their composition can be classified as a component model.

In the next section, we provide a very brief overview of some component models and their main characteristics. The list is not complete, and can be increased by time. It should be understood as a provision of some characteristic examples, or examples of widely used component models in Software Engineering.

The AUTomotive Open System Architecture (AUTOSAR) [14], is the result of the partnership between several manufacturers and suppliers from the automotive field. It envisions the conception of an open standardized architecture aiming at improving the exchangeability of diverse elements between vehicle platforms, manufacturer's applications and supplier's solutions. Those objectives rely upon the utilisation of both a component-based approach for the application and standardized layered architecture. This allows separating the component-based application from the underlying platform. AUTOSAR support both the client-server and Sender-Receiver communication paradigms and each AUTOSAR Software Component instance from a vehicle platform is only assigned to one Electronic Control Unit (ECU). The AUTOSAR Software Components, as well as all the modules in an ECU, are implemented in C.

BIP [14] framework designed at Verimag for modelling heterogeneous real-time components. This heterogeneity is considered for components having different synchronization mechanisms (broadcast/rendez-vous), timed components or non-timed components. Moreover, BIP focuses more on component behaviours than others component models thanks to a three-layer structure of the components (Behaviour, Interaction, Priority); a component can be seen as a point in this three-dimensional space constituted by each layer. This also sets up the basis for a clear separation between behaviour and structure. In this model, compound components, i.e components created from already existing ones, and systems are obtained by a sequence of formal transformations in each of the dimension. BIP comes up with its own programming language but targets C/C++ execution. Some

connections to the analysis tools of the IF-toolset [16] and the PROMETHEUS tools [17] are also provided.

CORBA Component Model (CCM) [18] evolved from Corba object model and it was introduced as a basic model of the OMG's component specification i.e CORBA 3 in 2002. The CCM specification defines an abstract model, a programming model, a packaging model, a deployment model, an execution model and a metamodel. The metamodel defines the concepts and the relationships of the other models. Component is a new CORBA metatype. CORBA components communicate with outside world through ports. CCM uses a separate language for the component specification: Interface Definition Language (IDL). CCM provides a Component Implementation Framework (CIF) which relies on Component Implementation Definition Language (CIDL) and describes how functional and non-functional part of a component should interact with each other. In addition, CCM uses XML descriptors for specifying information about packaging and deployment. Furthermore, CCM has an assembly descriptor which contains metadata about how two or more components can be composed together.

The Entreprise JavaBeans (EJB) [19], developed by Sun MicroSystems envisions the construction of object-oriented and distributed business applications in trying to hide to developers the underlying complexity, such as transactions, persistence, concurrency, interoperability. It also aims at the improvement of component reusability in providing different utilities, such as means, so called EJB-jars to package components, called beans. Three different types of components coexist to match the specific needs of different applications (The EntityBeans the SessionBean and the MessageDrivenBeans). Each of these beans is deployed in an EJB Container which is in charge of their management at runtime (start, stop, passivation or activation). In order to achieve this, EJB technology use the Java programming language.

Fractal [20] is a component model developed by France Telecom R&D and INRIA. It intends to cover the whole development lifecycle (design, implementation, deployment and maintenance/management) of complex software systems. It comes up with several features, such as nesting, sharing of components and reflexivity in that sense that a component may respectively be created from other components, be shared between components and describes its own behaviour. The main purpose of Fractal is to provide an extensible, open and general component model that can be tuned to fit a large variety of applications and domains. Consequently, nothing is fixed in Fractal; On the contrary, it even provides means to facilitate adaptation in notably having different implementations to fit the specific needs of a domain as for example its C-implementation called Think, which targets especially the embedded systems. A reference implementation, called Julia and written in Java, is also provided. Fractal can also be seen as a generic component model which intends to encompass other component models.

Koala [21] is a component model developed by Philips for building consumer electronics. Koala components are units of design, development and reuse. Semantically, components in Koala are defined in a ADL-like language. Koala IDL is used to specify Koala component interfaces, its Component Definition Language (CDL) is used to define Koala components, and Koala Data Definition Language (DDL) is used to specify local data of components. Koala components communicate with their environment or other components only through explicit interfaces statically connected at design time. Koala targets C as implementation language and uses source code components with simple interaction model.

Microsoft Component Object Model (COM) [22] is one of the most commonly used software component models for desktop and server side applications. A key principle of COM is that interfaces are specified separately from both the components that implement them and those that use them. COM defines a dialect of the Interface Definition Language (IDL) that is used to specify object-oriented interfaces. Interfaces are object-oriented in the sense that their operations are to be implemented by a class and passed a reference to a particular instance of that class when invoked. A concept known as interface navigation makes it possible for the user to obtain a pointer to every interface supported by the object. This is based on VTable. Although COM is primarily used as a general-purpose component model it has been ported for embedded software development.

The Open Services Gateway Initiative (OSGi) [23] is a consortium of numerous industrial partners working together to define a service-oriented framework with an "open specifications for the delivery of multiple services over wide area networks to local networks and devices". Contrary to most component definitions, OSGI emphasis the distinction between a unit a composition and a unit of deployment in calling a component respectively service or bundle. It offers also, contrary to most component models, a flexible architecture of systems that can dynamically evolve during execution time. This implies that in the system, any components can be added, removed or modified at run-time. Thus, there is no guaranty that a service provided at a certain time will be still provided later. Being built on Java, OSGI is platform independent.

Pecos [24] is a joint project between ABB Corporate Research and academia. Their goal is to provide environment that supports specification, composition, configuration checking and deployment for reactive embedded systems built from software components. Component specification and component composition are done in an ADL-like language called CoCo. There are two types of components, leaf components and composite components. The inputs and outputs of a component are represented as ports. At design phase composite components are made by linking their ports with connectors. Pecos targets C++ or Java as implementation language, so the run-time environment in the deployment phase is the one for Java or C++. Pecos enables specification of EFP properties such as timing and memory usage in order to investigate in prediction of the behaviour of embedded systems.

Pin [25] component model is based on an earlier component technology developed by Carnegie Mellon Software Engineering Institute (SEI), for use in prediction-enabled component technologies (PECTs). It is aimed for building embedded software applications. By using principles from PECT it aims at achieving predictability by construction. Components are defined in an ADL-like language, in the "component and connector style", so called Construction and Composition Language (CCL). Furthermore, Pin components are fully encapsulated, so the only communication channels from a component to its environment and back are its pins.

Robocop [26] is a component model developed by the consortium of the Robocop ITEA project, inspired by COM, CORBA and Koala component models. It aims at covering all the aspects of the component-based development process for the high-volume consumer device domain. A Robocop component is a set of possibly related models and each model provides particular type of information about the component. The functional model describes the functionality of the component, whereas the extra-functional models include modelling of timeliness, reliability, safety, security, memory consumption, etc. Robocop components offer functionality through a set of 'services' and each service may define several interfaces. Interface definitions are specified in a Robocop Interface Definition Language (RIDL). The components can be composed of several models, and a composition of components is called an application. The Robocop component model is a major source for ISO standard ISO/IEC 23004 for multimedia middleware.

Rubus [27] component was developed as a joint project between Arcticus Systems AB and the Department of Computer Engineering at Mälardalen University. The Rubus component model runs on top of the Rubus real-time operating system. It focuses on the real-time properties and is intended for small resource constrained embedded systems. Components are implemented as C functions performed as tasks. A component specifies a set of input and output ports, behaviour and a persistent state, timing requirements such as release-time, deadline. Components can be combined to form a larger component which is a logical composition of one or more components.

SaveCCM [28], developed within the SAVE project and several Swedish Universities, is a component model specifically designed for embedded control applications in the automotive domain with the main objective of providing predictable vehicular systems. SaveCCM is a simple model that constrains the flexibility of the system in order to improve the analysability of the dependability and of the real-time properties. The model takes into consideration the resource usage, and provides a lightweight run-time framework. For component and system specification SaveCCM uses "SaveCCM language" which is based on a textual XML-syntax and on a subset of UML2.0 component diagrams.

The SOFA (Software Appliances) [29] is component model developed at Charles University in Prague. A SOFA component is specified by its frame and architecture. The frame can be viewed as a black box and it defines the provided and required interfaces and its properties. However a framework can also be an assembly of components, i.e a composite component. The architecture is defined as a grey-box view of a component, as it describes the structure of a component until the first level of nesting in the component hierarchy. SOFA components and systems are specified by an ADL-like language. Component Description Language (CDL). The resulting CDL is compiled by a SOFA CDL compiler to their implementation in a programming language C++ or JAVA. SOFA components can be composed by method calls through connectors. The SOFA 2.0 component model is an extension of the SOFA component model with several new services: dynamic reconfiguration, control interfaces and multiple communication styles between the components.

4. COMPONENT MODEL CLASSIFICATION

In order to illustrate the utilization of our classification framework, we categorize here the component models listed above with respect to the corresponding dimensions. The reference documentation of each component models has generally been used to fill those tables. However, some of the information presented here are not mentioned explicitly in the reference documentation and are subject to the reader's point of view.

Table 1: Lifecycle Dimension

Component Models	Modelling	Implementation	Packaging	Deployment
AUTOSAR	N/A	C	N/A	At compilation
BIP	A 3-layered representation: statemachine diagram, priority and interaction expression or a statemachine with ports	BIP language	N/A	At compilation
CCM	Abtstract model:OMG-IDL, Programming model: CIDL	Language independent.	Deployment Unit archive (JARs,DLLs)	At run-time
Fractal	FractalGui, ADL-like language (Fractal ADL, Fractal IDL), Annotations (Fractlet)	Julia, Aokell(Java) Think(C/C++) FracNet(.Net) …	File system based repository	At run-time
KOALA	ADL-like language (IDL,CDL and DDL)	C	File system based repository	At compilation
EJB	N/A	Java, Java binary code	EJB-Jar files	At run-time
MS COM	Microsoft IDL	Different languages, Binary standard	DLL	At run-time
OSGi	N/A	Java	Jar-files (bundles)	At run-time
PIN	ADL-like language (CCL)	C	DLL	At compilation

Component Models	Modelling	Implementation	Packaging	Deployment
PECOS	ADL-like language (CoCo)	C++, Java	Jar packages	At compilation
ROBOCOP	IDL for the interface model. Several different models	C, C++	zip files	At compilation At run-time
RUBUS	N/A	C	File system based repository	At compilation
SaveCCM	ADL-like language	C	N/A	At compilation
SOFA 2.0	Meta-model based definition	Java	Repository	At run-time

Table 2: Constructs

Component Models	Interface Specification				Interface Levels	Standard Architecture Styles	Communication Type	Binding Type	
	Interface/ Ports/ Both	Distinction Provides / Requires	Distinctive feature	Interface Language				Exogenous	Vertical
AUTOSAR	Both	Yes	Classified within 3 types: – AUTOSAR Interface, – Standardized AUTOSAR Interface, – Standardized Interface	C header files	Syntactic No semantic No behaviour	Client/Server Data-centered	Synchronous Asynchronous	No	Delegation
BIP	Port	No	Existence of: Complete interfaces, Incomplete interfaces	BIP Language	No syntactic Semantic Behaviour	Event-driven	Synchronous Asynchronous (Rendez-vous, Broadcast)	No	Delegation
CCM	Both	Yes	Classified within 2 types: – Facets and receptacles – Event sinks and event sources	CORBA IDL	Syntactic No semantic No behaviour	Blackboard	Synchronous Asynchronous	No	No
EJB 3.0	Interface	No	N/A	Java Programming Language + Annotations	Syntactic No semantic No behaviour	Client/Server (JDBC) Blackboard (JMS)	Synchronous Asynchronous	No	No
Fractal	Interface	Yes	Existence of: Control Interface	Any programming language, IDL, Fractal ADL	Syntactic No semantic Behaviour	Multiple architectural styles	Multiple communication styles	Yes	Aggregation
KOALA	Interface	Yes	Existence of: – Diversity Interface, – Optional Interface	IDL	Syntactic No semantic No behaviour	Pipe&filter	Synchronous	Yes	Aggregation
MS COM	Interface	No	All interfaces derived from a IUnknown	Microsoft IDL	Syntactic No semantic No behaviour	Multiple architectural styles	Synchronous	No	Delegation Aggregation
OSGi	Interface	Yes	Existence of: Dynamic Interfaces	Java programming language	Syntactic No semantic No behaviour	Event-driven	Synchronous	No	No
PECOS	Port	Yes	N/A	Coco composition language	Syntactic Semantic Behaviour	Pipe&filter (with blackboard) Event-driven	Synchronous	No	Delegation
Pin	Port	Yes	N/A	Component Composition Language	Syntactic No semantic No behaviour	Pipe&filter Event-driven	Synchronous Asynchronous	Yes	No

Rubus	Port	Yes	Classified within 2 types: – data – triggered	C header file	Syntactic No semantic No behaviour	Pipe&filter	Synchronous	No	Delegation
Robocop	Port	Yes	N/A	Robocop IDL (RIDL)	Syntactic Semantic (Behaviour)	Client/Server	Synchronous Asynchronous	No	No
SaveCCM	Port	No – but input/output interface	Classified within 3 types: – data – triggered – data& triggered	SaveComp (XML-based)	Syntactic No semantic Behaviour	Pipe&filter	Synchronous	No	Delegation
SOFA 2.0	Interface	Yes	Existence of: Utility Interface, Possibility to annotate interface and to control evolution	Java programming language	Syntactic No semantic Behaviour	Multiple architectural styles	Multiple communication styles	Yes	Delegation

Table 3: Extra-Functional Properties

Component Models	General support for properties	Properties specification	Composition support
AUTOSAR	N/A		
BIP	Behaviour compositions, endogenous EFP management	times properties	
CCM	Support for mechanism to influence of some EFP, exogenous EFP management	N/A	Run time support for some EFP
Fractal	Interceptor and controller in Julia, extension possibilities for different EFP, endogenous EFP management	Extension with a new controller	
KOALA	Extensions in interface specification and the compiler	Resource usage but no timing and memory consumption	Compile time checks
EJB	Support for mechanism to influence of some EFP, container maintaining EFP, exogenous EFP management	N/A	Run time support for some EFP
MS COM	Endogenous EFP management	N/A	N/A
OSGi	endogenous EFP management	N/A	N/A
PIN	Analytic interface for EFP, endogenous EFP management	Timing properties	Different EFP composition theories
PECOS	endogenous EFP management	Timing properties (WCET, periods), memory consumption	N/A
ROBOCOP	CEP provides manager for resource budgets, exogenous EFP management	Memory consumption, WCET, cycle time, priority, reliability	Some EFP checked at deployment and monitored during execution.
RUBUS	Exogenous EFP management	Timing, resource usage, QoS	Design time
SaveCCM	Interface extension endogenous EFP management	Real-time attributes	Composition of RT EFP
SOFA 2.0	Behaviour EFP specification, endogenous EFP management	Behavioural (protocols)	Composition at design

Table 4: Domains

Domain	AUTOSAR	BIP	CCM	Fractal	KOALA	EJB	MS COM	OSGi	PIN	PECOS	ROBOCOP	RUBUS	SaveCCM	SOFA 2.0
Generative				x							x			x
General			x	x		x	x	x	x					x
Specialised	x	x			x			x		x	x	x	x	

5. CONCLUSION

In this short survey we have presented a framework for classification of component models. Such classification is not simple, since it does not cover all aspects of component models. It however identifies the minimal criteria for assuming a model to be a component model and it groups the basic characteristics of the models. From the results we can recognize some recurrent patterns such as – general-purpose component models utilize client-server style, while in the specialized domains (mostly embedded systems) pipe & filter is the predominate style. We can also observe that support for composition of extra-functional properties is rather scarce. There are many reasons for that: in practice explicit reasoning and predictability of EFP is still not widespread, there is an unlimited number of different EFPs, and finally the compositions of many EFPs are not only the results of component properties, but also the context in which they are composed [12]. This taxonomy can be further analyzed and refined, which is our intention: on one side enlarge the list with the new component models, on other side refine the taxonomy by introducing some comparative values and by introducing subtypes of the points in the framework dimension.

6. REFERENCES

[1] C. Szyperski, *Component Software*, Addison-Wesley Professional; 2002

[2] G. T. Heineman and W. T. Councill, Component-based Software Engineering: Putting the Pieces Together, Addison-Wesley, 2001.

[3] M. R. V. Chaudron, *Lecture notes on CBSE* Technische Universiteit Eindhoven, 2006

[4] N. Medvidovic, E. M. Dashofy, R. N. Taylor, *Moving architectural description from under the technology lamppost*, Information and Software Technology,vol.49, Iss.1, 2007

[5] N. Medvidovic and Ri. N. Taylor, *A Classification and Comparison Framework for Software Architecture Description Languages*, IEEE Transactions on Software Engineering, Vol. 26, No. 1, January, 2000

[6] K.-K. Lau and Z. Wang. *A taxonomy of software component models*. In Proc. 31st Euromicro, Conference, pages 88–95. IEEE Computer Society Press, 2005.

[7] G. Kotonya, I. Sommerville and S. Hall, Towards *A Classification Model for Component-Based Software Engineering Research*, Proc. of the IEEE 29th EUROMICRO Conference, September 2003

[8] I. Crnkovic, M. Chaudron, S. Larsson *Component-based Development Process and Component Lifecycle,* Journal of Computing and Information Technology, vol 13, nr 4, 2005

[9] I. Crnkovic, M. Larsson, *Building Reliable Component-Based Software* Systems Artech House, 2002

[10] A. Beugnard, J.-M. Jézéquel, and N. Plouzeau. *Making components contract aware*. IEEE Computer, 32(7):38-45, 1999.

[11] Miller, G. A. (2002). *WordNet*®. Cognitive Science Laboratory, Princeton University Available: http://www.cogsci.princeton.edu/~wn/

[12] I. Crnkovic, M. Larsson, O. Preiss, *Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes*, Architecting Dependable Systems III,, p pp. 257 – 278, Springer, LNCS 3549, 2005

[13] The Object Management Group, *UML Superstructure Specification* v2.1, April 2006.
Available at http://www.omg.org/docs/ptc/06-04-02.pdf

[14] AUTOSAR Development Partnership, *AUTOSAR – Technical Overview v2.0.1*, 27/06/2006,
Available at
http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf

[15] Ananda Basu, Marius Bozga and Joseph Sifakis, *Modeling Heterogeneous Real-time Components in BIP*, 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM06), Invited talk, September 11-15, 2006, Pune, pp 3-1

[16] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, *The IF toolset*, in SFM, 2004.

[17] G. Gößler, *PROMETHEUS — a compositional modelling tool for real-time systems*, Proc. Workshop RT-TOOLS 2001, Technical report 2001-014, Uppsala University

[18] OMG CORBA v 4.0, http://www.omg.org/docs/formal/06-04-01.pdf

[19] EJB 3.0 Expert Group, JSR 220: Enterprise JavaBeansTM,Version 3.0 EJB Core Contracts and Requirements Version 3.0, Final Release, May 2, 2006.,

[20] E. Bruneton, T. Coupaye & J.B. Stefani, *The Fractal Component Model*, February 5, 2004.
http://fractal.objectweb.org/specification/index.html

[21] R. van Ommering, F. van der Linden, and J. Kramer. *"The koala component model for consumer electronics software",* In IEEE Computer, pages 78–85. IEEE, March 2000.

[22] D. Box, Essential COM, Addison-Wesley Professional, 1997

[23] OSGi Alliance, 15/02/2007, http://www.osgi.org/

[24] M. Winter, C. Zeidler, C. Stich, *"The PECOS Software Process"*, Workshop on Components-based Software Development Processes, ICSR 7 2002.

[25] S. Hissam, J. Ivers, D. Plakosh, K. Wallnau, *Pin Component Technology (V1.0) and Its C Interface*. CMU Technical Report, CMU/SEI-2005-TN-001

[26] H. Maaskant; "A Robust Component Model for Consumer Electronic Products", Philips Research Book Series Volume3, p167-192

[27] Arcticus Systems, *Rubus component model*, http://www.arcticus-systems.com

[28] M. Åkerholm et al., *The SAVE approach to component-based development of vehicular systems*, Journal of Systems and Software, Elsevier, May, 2006

[29] T. Bureš, P. Hnětynkal and F. Plášil, *SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model*, Proc. of SERA 2006, Seattle, USA, IEEE CS, Aug 2006

[30] Oxford Advanced Learner's Dictionary,
http://www.oup.com/oald-bin/web_getald7index1a.pl

Usability Patterns in Design of End-user Tailorable Software

Jeanette Eriksson
Blekinge Institute of Technology
P.O. Box 520
S-37225 Ronneby
+46 457 358000
jeanette.eriksson@bth.se

ABSTRACT
Design of end-user tailorable software requires a cooperative design process were users and developers participate on equal terms. Tailoring differ from other interactive software in that end-users continue to evolve the software in use time. Users are co-designers. To be able to fruitfully work together users and developers has to reach mutual understanding. The objective for this paper is to provide such common ground by adopting patterns to end-user tailoring. Different types of patterns can act as a mediating artefact between users and developers and as usability is close to the user domain, usability pattern can act as a gateway to other types of patterns in the architectural design process. To facilitate the work and introduction of pattern in a cooperative design project we make a selection of usability patterns that are of vital importance for the success of end-user tailorable software, and also have architectural impact and therefore should be addressed early in the design process. The result is a small collection of usability scenarios with corresponding usability patterns that are especially important to tailoring. The usability patterns are related to different types of tailoring through an existing categorization. A comparison between different pattern structures is also presented and resulted in a pattern template suitable for the cooperative design process of end-user tailoring.

Keywords
Cooperative design, usability pattern, architecture, end-user tailoring

1. INTRODUCTION
In a fast changing world more and more flexibility is needed in software to supply support for higher reusability and prevent the software from expiring too fast. “Real-world systems must change or they die” [15, s. 22]. One way to provide this kind of flexibility is end-user tailoring. A tailorable system is modified while it is being used as opposed to changed during the development process. To tailor a system is to “continuing designing in use” [12] It is possible for the user to change a tailorable system by support of some kind of interface.

Tailorable software is needed when the environment is characterized by fast and continuous change. As Stevens and his colleagues put it “The situatedness of the use and the dynamics of the environment make it necessary to build tailorable systems. However, at the same time these facts make it so difficult to provide the right dimensions of tailorability.” [22]. This paper aims for providing support for the process of designing end-user tailorable software by introducing patterns as a mediation artefact between users and developers.

The development of tailorable software is an ongoing process where users are co-designers [8] as it is users that evolve the software in use time. The absence of end-user participation can result in low acceptance of the software [21] and in end-user tailoring user acceptance is especially important as it is the users that carry out the intension with the software, to be evolved. We agree with [13] that the users’ view of the system is not only the interface. Task related needs are what motivate end users to make changes to the system [18].

As the users are co-designers human-centered design are required when designing tailorable software. The users bring profound knowledge of the business process and organizational issues into the development project, that should be made use of in the design of the technical solution [11]. But it is difficult to get active involvement of the end-users in the development process [21]. This is confirmed by our own interviews with users and developers in a Swedish telecom company. Both users and developers express their desire and an interest in achieving an environment were users and developers take active part and equal responsibility of the software developed, but they also agree on that it is not easy to achieve. A prerequisite to make such a cooperative process to work is that users and developers share the same language [20]. Or in other words they have to reach a mutual understanding.

A classification can be a useful tool to understand a phenomenon as tailoring. A classification of tailoring consisting of four categorises of tailoring is presented in [6]. The categorization is designed to take both user and system perspective into account so that the categorization can act as a base for communication between developers and user when designing tailorable software. The categorization is intended as a means of communications to involve the users more in the design process and was found promising for use in industry. The categorization is presented in Section 2.

Another obstacle to overcome is the knowledge transfer of technical issues from developers to users. This is a difficult matter, but patterns have been found to be a useful tool [17, 20, 21] of knowledge transfer. Patterns facilitate understanding and communication, increase confidence in decisions, make it easier to consider different solutions and provide for control [4].

What we need to use a pattern approach in end-user tailoring design is a selection of suitable patterns. This selection of patterns should be connected to the categorization of tailoring to be able to narrow down the numbers of patterns to consider for each type of tailoring. Especially for beginners it is hard to have a lot of patterns to consider [10]. As we believe that end-user participation

in the design process is essential to gain quality in end-user tailorable software it is important to neutralize possible obstacles.

There are two ways to introduce patterns in the cooperative design process, either by starting with architectural design patterns that transfer good practice when it comes to architectural design or patterns that expresses design issues of interaction or HCI (Human Computer Interaction). We call such patterns usability patterns. The content of usability patterns is closely related to the task and the users' domain and usability patterns may provide for a gentle slope into patterns for software architectures. Therefore it is preferable to start with using usability patterns. Usability patterns do not only deal with issues that are put on top of the basic software architecture. In fact separation of concern is not enough to achieve usability [14]. Usability features that are recognized late in the design process are often expensive to attend to. Usability issues obviously have architectural impact beyond the detailed design of graphical interfaces and several usability scenarios are identified to influence software architecture [2]. This paper focus on usability patterns with architectural impact, that are of vital importance to end-user tailoring.

In summary we have two research questions to answer:

- What usability patterns with architectural impact should initially be introduced and how are they related to the different categories of end-user tailoring?
- What should a pattern consist of to be supportive in the cooperative design process involving both user and developers?

The result is a classification of patterns that can act as a mediating media between users and developers as well as a concrete base for the technical solution when designing end-user tailorable software.

In the end-user tailoring community patterns are not discussed very frequently. It is likely that the researchers and practitioners that are in the area of end-user tailoring use patterns, but there is not a explicit discussion of the topic in the research community. We therefore argue that there is a need to classify patterns suitable for end-user tailorable software not only from an industrial perspective but also from an academic point of view as patterns seems to be a kind of blind spot in the area of end-user tailoring. We do not claim the collection of patterns presented in this article is exhaustive. We indeed hope the collection will be extended with more dedicated patterns.

The rest of the paper is structured as follows. The next section will present the categorization of tailoring. Section 3 describes two approaches to identify usability patterns that have to be introduced early in the development. In Section 4 the usability patterns of vital importance to end-user tailoring is explored and presented. Section 5 contains a description of how the patterns for end-user tailoring can be presented. Thereafter follows a discussion of how the results relate to other work and how the results can be used. The paper ends with a summary of the results.

2. CATEGORIZATION OF TAILORING

The categorization proposed by Eriksson et al.[6] is intended as a means of communication between developers and users in situations when deciding what kind of tailorability to implement. The categorization takes into account both a user perspective and a system perspective. The user perspective represent what changes can be done or the intention with the activity, while the system perspective corresponds to how the change is achieved in the system (on a high level). The categorization is shown in Table 1.

Table 1. Categorization of tailorable software

Category	User Perspective	System Perspective
Customization	Small changes (set of parameter values)	Parameter Values are interpreted and used in existing code.
Composition	Link different existing components	The relationships between the components are defined by a composition language. (It does not matter which programming language)
Expansion	Creation of a new component.	Components are integrated into the software by the implementation language and the new component does not differ from the pre-existing components. The composed component is used as a starting point for further tailoring. The software may generate code that is added to the pre-existing code or in another way incorporate the new component into the application.
Extension	Insertion of code.	New code (implemented by the end-user) is added to the pre-existing code. The application may also generate code to integrate the end-user's code to the software.

Customization is the simplest way of doing tailoring. It means that the user sets some values on one or more parameters and those parameters manage what functionality that is used. *Composition* means that the user has a set of components to choose from and he or she can connect them in specific ways to reach the desired functionality. *Expansion* also mean that the user chooses components out of a set, but the difference is that the users' combination of components are build into the system to be an integrated part. The new component is treated as the predefined components and will be accessible in the set to

choose from next time the software is tailored. *Extension* is the category which provides for the highest flexibility. It means that the user writes code that is integrated into the system either by wrapping up the new code into system generated code or, if written in a predefined way, just adding it to the code mass of the software. The user can either write the code in some high level language or some visual programming language.

The categorization can be used as a gateway to which patterns to consider. By defining both a user and a system perspective, the intention is to make it easier to discuss tailoring in a consistent way. As the usability patterns deals with the user perspective we will only use this perspective on tailoring in the following discussion.

3. USABILITY PATTERNS

Usability patterns or HCI (Human Computer Interaction) design patterns are useful tools when designing user interfaces [28]. And there exists a number of different collections of patterns, for example a comprehensive pattern language for user interfaces by Tidwell [24, 25, 27]. Traditionally HCI (interface design) and software architectures have been kept separate by the notion of separation of concerns, but separation of concerns are not enough if we want to design software with good usability, highly accepted by users. Usability issues discovered late in the process can be expensive to recover [14] which implicate that usability issues have impact on an architectural level of software design. There are two recent approaches (presented below) that deals with usability issues that should be considered early in the design process.

Based on experience Bass and John [2] have identified 27 usability scenarios that have to be considered during the architectural design. For each scenario they created an architectural pattern as a solution to the scenario. The 27 scenarios are in short:

1. Aggregating data
2. Aggregating commands
3. Cancelling commands
4. Using applications concurrently
5. Checking for correctness
6. Maintaining device independence
7. Evaluating the system
8. Recovering from failure
9. Retrieving forgotten passwords
10. Providing good help
11. Reusing information
12. Supporting international use
13. Leveraging human knowledge
14. Modifying interfaces
15. Supporting multiple activities
16. Navigating within a single view
17. Observing system state
18. Working at the users' pace
19. Predicting task duration
20. Supporting comprehensive searching
21. Supporting undo
22. Working in an unfamiliar context
23. Verifying recourses
24. Operating consistently across views
25. Making views accessible
26. Support visualization
27. Supporting personalization

A similar attempt to introduce usability aspects early in the development process was done within a European Union project (STATUS) [7, 9, 16]. But they started from a different angle compared to Bass and John. The STATUS project started out with a set of usability attributes (satisfaction, learnability, efficiency and reliability) and then mapped the attributes to usability properties that in its turn were related to usability patterns. A usability property is specified in terms of the solution space and can be regarded as usability requirements expressed in a more concrete form. For example the quality attribute *efficiency* has a relation to the usability property *error prevention* as error prevention has a positive effect on efficiency. Error prevention in turn has a relation to, for example, the usability patterns *form or field validation* and *workflow model* [16] as the patterns fulfil the requirement.

The results from the two approaches are overlapping and consist of a set of usability pattern that have an impact on software architecture and thus has to be considered early in the development process. The relationship between the usability patterns from the STATUS project and the general usability scenarios provided by Bass and John is presented in [16].

4. PATTERNS FOR TAILORING

Our goal is to match the categories of end-user tailoring to a set of usability patterns that are especially important to provide for user satisfaction and confidence in the tailoring process. To achieve this we have made use of both approaches presented in Section 3.

During a project done in corporation with our industrial partner, a major telecom operator, we explored how end-users could manage system infrastructure. We built a prototype that was evaluated by users and developers by "talking aloud" when using the prototype. In the same project we explored what technical issues that are most important to consider to make end-user tailoring work. Four usability issues or overall requirements were revealed concerning the tailoring interface [5]:

1. Functionality for controlling and testing
2. Clear split between definition, execution and the tailoring process.
3. Unanticipated use revealed to the tailor.
4. Complexity

Functionality for controlling and testing is self-explanatory. It is essential that the user can control the tailoring process and test the changes. It was also important for the users to have a clear

split between use and tailoring. One reason was that it was easier to focus on one abstraction level at a time and another reason was that a clear split makes it possible to assign different people for the different tasks. In other words it is easier to separate the role 'tailor' from the role 'user' and thereby delegate the tailoring process to a few people. It was also evident that it was important that the different possibilities to change the software was revealed to the tailors even though it might not be what the designers had in mind when designing the tailoring feature. The software should be prepared for creative use. The last issue concerning complexity is somewhat connected to unanticipated use and it was shown that the users preferred a more complex tailoring interface with superfluous information in favour of just in time information to minimize cognitive load that are advocated as a pattern to support. The motivation was that a tailoring activity is not performed at a regular basis and therefore it is allowed to take time. It is therefore preferable to have a complex interface that allow for creative use. But a complex tailoring interface requires a very simple user interface in compensation. As the complexity issue is diametrically to what is recommended in usability literature, we will not discuss complexity further. We do not need a pattern to decrease the complexity. But for the interested reader there are patterns to handle complex data in user interfaces [25, 26].

The first step towards a match between usability patterns and the tailoring categories is to match the usability issues presented above (unanticipated use revealed to the tailor, explicit user control, error correction and error prevention) with usability properties. The usability issues are requirements on end-user tailorable software and correspond well to usability properties as the properties also are a form of requirements. Then the usability issues are mapped to the general usability scenarios. For example, if an end-user tailorable system provide for *unanticipated use revealed to the tailor* it also has to provide for the usability properties *explicit user control, error correction and error prevention* [9]. Then we examine the general usability scenarios. If you fulfil the requirement of error prevention it is easier to work in an unfamiliar context. Likewise if you fulfil the requirement of guidance you provide for good help. The summary of the correspondence is shown in Table 2.

Table 2 results in a subset of scenarios that are of vital importance to end-user tailoring, Scenarios corresponds to activities and it is therefore convenient to match the subset of scenarios to the categories of tailoring as the describe how the changes are performed. It is therefore easy to imagine what scenarios should be relevant for the different categories. For example, independent of what kind of tailoring activity you perform you would like to be able to check for correctness, support of undo and good help. But if you do a composition, combine different component to each other, it involves doing things you are not doing on a regular basis. The scenario of *working in an unfamiliar context* is equivalent with what you are doing. The match is presented in Table 3. Then the relationship between the categories reveal themselves automatically by matching the scenarios with usability patterns according to [16] (Table 3).

The result is a selection of usability patterns that have an architectural impact. By choosing a type of tailoring to implement we get some examples of usability patterns we should consider to use. We do not claim that the selection is complete. Actually there may be other usability patterns that match the scenarios and should be considered to be used. Note that we have made a selection of usability scenarios that are of vital importance, thereby we do not say that the rest is unimportant for end-user tailoring. In the contrary those scenarios with corresponding usability patterns are as important to tailorable software as to any other software concerned with user interaction. The rest of the scenarios can be used as a checklist to determine if important usability issues have been considered during architectural design. What we say is that the selected scenarios are not negotiable if the end-user tailorable software is to be a success. For example providing for good help is not negotiable and therefore one of the patterns "Wizard", "Context-sensitive help", "Standard Help" or "Tour" should be considered.

Table 2. Relations between usability issues and properties.

Usability issue	**Usability property [9]**	**Usability scenario [2]**
Functionality for controlling and testing	Explicit user control Error management • Error correction • Error prevention	Checking for correctness Observing system state Supporting undo Working in an unfamiliar context Verifying resources
clear split between definition, execution and the tailoring process.	Adaptability • Matching user preferences • Matching user expertise	
unanticipated use revealed to the tailor.	Guidance Provide feedback	Providing good help Providing good help

5. PATTERN STRUCTURE

It is important that the different patterns are not too comprehensive. One intention with the patterns is that both users and developers may get an overview of the different design possibilities. To make the patterns easy to understand by end-users it is essential that they are written in a more prosaic style than if the patterns are solely intended for the developers to use [21]. The patterns should provide the participants with an understanding of the pattern almost *at a glance* at the same time it is essential that the patterns provide the participants, both users and developers with enough information to be able to transform the pattern into the software architecture without having to invent the wheel once again. In other words the patterns should not only be a base of discussion but at the same time also an effective tool for the developers.

Table 3 Tailoring categories and corresponding scenarios and pattern.

Category	Usability Scenario	Pattern [16]
Customization	Checking for correctness Supporting undo Providing good help	Form/Field validation Undo Wizard, Context-sensitive help, Standard Help, Tour User profile
Composition	Checking for correctness Supporting undo Providing good help Working in an unfamiliar context	Form/Field validation Undo Wizard, Context-sensitive help, Standard Help, Tour User profile Workflow model
Expansion	Checking for correctness Supporting undo Providing good help Working in an unfamiliar context Observing system state	Form/Field validation Undo Wizard, Context-sensitive help, Standard Help, Tour User profile Workflow model Status indication
Extension	Checking for correctness Supporting undo Providing good help Working in an unfamiliar context Observing system state Verifying resources	Form/Field validation Undo Wizard, Context-sensitive help, Standard Help, Tour User profile Workflow model Status indication Alert

There is a lot of different pattern forms [4]. We have chosen to compare four different approaches to evaluate the suitability of using one of the approaches for the patterns aimed for end-user tailoring and determine if we should put together our own pattern template. The four approaches are chosen because they fulfil at least one of the requirements for a pattern template for end-user tailoring. Borchers pattern structure [3] is uniform and supports both application domain patterns, HCI patterns and software patterns, which is an advantage as we, in the future, want to incorporate software design patterns in the design process as well. Schümmer's et.al. [17, 21] supports both users and developers and are constructed as a means of communication, which is exactly what we also want to do. John et. al. [14] explicitly manifest the importance of consider different types of forces influencing the design, which we consider important, and the last approach is Gamma's et.al [10] which is the most widely known pattern collection. This collection is written for developers and since an end-user tailoring pattern also should be useful and effective for developers when implementing the software, it is relevant to compare the other approaches with this.

Borchers [3] extends the notion of pattern languages in to Human-Computer Interaction as patterns is a suitable tool to capture experiences of user interface design. Borchers also extends the pattern language approach to the area of the application domain. Borchers has worked a lot with interactive exhibitions in, for example, music. Borchers has constructed a interdisciplinary pattern language framework to be able to collect design experiences from both HCI, software engineering and the application domain. The pattern structure is uniform and is intended to be suitable for all three areas. Table 4, left column, lists the different subsections in the pattern structure.

Schümmer and colleagues [17, 21] outline a pattern structure of design patterns that are constructed to meet both users' and developers' requirements of detailed description and visualization. This structure was tried out in two projects and found useful in the context of educational groupware. The patterns acted as metaphors and made it possible for the participants to talk about the software system and it also helped the participants to focus on one feature at a time [21]. The pattern structure is used for a pattern language and is constructed to facilitate communication and learning. The pattern template consists of three main sections. The first section is to facilitate deciding if the patterns seams to fit the situations, the second section contains solutions and the last part present the solution in more detail. Table 4, second column, lists the different subsections in the pattern structure.

Most patterns, design as usability pattern are constructed as the pattern are independent of external forces [14], but John et.al [14] have constructed a structure for usability-supporting patterns that have a section dedicated for a 'Specific Solution'. John et.al have identified different types of forces that influence the implementation of the patterns and incorporated them in their usability-supporting patterns. This makes the pattern dependent of the actual situation it would be used in. The forces identified are:

- Forces exerted by the environment and the task
- Forces exerted by human desires and capabilities
- Forces exerted by the state of the software
- Forces that come from prior design decisions

Table 4. Comparison of four different pattern structures

Pattern for an interdisciplinary framework [3]	**Pattern for user participation [17, 21]**	**Usability-supporting pattern [14]**	**Pattern by Gamma et.al [10]**
Pattern Name			
• Name	• Pattern name	• Name	• Pattern name
Problem			
• Ranking			• Also known as
• Illustration	• Intent		• Intent
• Context (relates to other patterns) and references	• Context	Usability Context • Situation: (usefulness from the end-users' perspective) • Conditions on the Situation (conditions of usefulness) • Potential Usability Benefits	
• Problems and forces	• Problem	Problem • Forces exerted by the environment and the task • Forces exerted by human desires and capabilities • Forces exerted by the state of the software	
• Illustration	• Scenario	•	• Motivation
• Illustration	• Symptoms (identify the need)	•	• Applicability
Solution			
• Solution	• Solution	• General solution: Responsibilities of the general solution that resolve the forces	
		Specific solution • Forces that come from prior design decisions • Allocation of responsibilities to specific components • Rational gives reason for how the responsibilities have been assigned to the components.	
• Diagram		• Component diagram of specific solution • Sequence diagram of specific solution • Deployment diagram of specific solution	• Structure
	• Collaboration		• Participants • Collaborations
Consequences			
	• Rationale		• Consequences
	• Danger spots (the raise of new unbalanced forces)		• Implementation
Extras			
			• Sample code
• Examples	• Known uses		• Known uses
• Context (above)	• Related patterns		• Related patterns

These identified forces correspond well to our own experiences from prolonged observations of a project developing an end-user tailorable subsystem of one of the business systems of telecom operator. Also Buschmann et.al [4] claim the forces are the heart of every pattern. Table 4, third column, lists the different subsections in the pattern structure.

Forth column in Table 4 lists the structure of Gamma et.al's patterns [10]. This approach is well known amongst developers and it is also developers that is the target group for the patterns. The patterns "help designers reuse successful designs by basing new designs on prior experience." [10, s. 1]. The patterns structure consists of not only graphical diagrams but also relationships between classes and objects, alternative solutions and trade-offs. Examples are also important as it shows how the pattern can be applied.

The question is which of the approaches that is most suitable for a pattern of end-user tailoring. We have to list the requirements for a pattern of end-user tailoring:

1. The pattern structure shall be practicable for software engineering design patterns too.
2. The patterns shall start generally and gradually be more detailed to facilitate learning.
3. The patterns shall be easy to overview, grasp and understand.
4. The pattern structure shall be an effective tool for both users and developers, together and individually.

If we compare how well the different approaches comply to the requirements (Table 4) we can see that Borchers' and Schümmer's et.al. approach is equally favourable. Borchers' pattern structure is better than Schümmer's et.al. when it comes to how practical it is for software engineering patterns, but it is compensated by the fact that Borchers' patterns are less detailed. It is easy to take care of the lack of details by adopting the parts from John's and Bass' approach, were the different forces are described in detail. John and Bass also recommend diagrams on a detailed level.

Table 5 Compliance of requirements (Legend: ++ = very good , + = good, - = not that good , -- = bad)

Requirement	**Pattern for an interdisciplinary framework [3]**	**Pattern for user participation [17, 21]**	**Usability-supporting pattern [14]**	**Pattern by Gamma et.al [10]**
Practical for SE patterns	+	-	++	++
Gradually more detailed	-	+	++	+
Easy to overview and understand	++	+	--	--
Good tool for both developers and users	+	+	-- user ++developer	-- user ++developer

Borchers pattern structure should be a good base and then fill in with good features from the other approaches. Borchers patterns start out in a general way and there are few headings, which makes it easier to grasp and overview. The headings are general and easy to understand. The details should not appear until further down in the solution part. The solution should first be introduced generally and then get more detailed. This is attended to by adding section *general solution* and *specific solution* from John's and Bass' pattern structure. But compared to patterns for user participation and Gamma's et.al pattern there are even more details that should be added to better support the developers. That is consequences, danger spots, sample code and Related Patterns. Related patterns are in Borcher's approach incorporated in the Context section. But we find it better to explicitly point out the related patterns, in favour of ease of use.

The resulting pattern structure is intended solely for end-user tailorable software and the tailoring categories act as a gateway to the patterns, therefore it is of course important to relate each pattern with the type of tailorability it is suitable for.

The template is constructed so that it begins in a general way and get more detailed and specialized further down. It is essential to remember that the descriptions in the pattern template have to be written in a way that complies with different types of stakeholders.

6. DISCUSSION

That design patterns are useful when designing software have been proven over and over again during the last decades. In 1997 when the design pattern concept in software engineering was intensely discussed Pree and Sikora [19] express their concern of design patterns being a hype, but now ten years later we are beyond the hype [4] an we can see that design patterns is here to stay. We have made an attempt to adjust a part of the patterns concept to end-user tailoring. Apart from the benefit from using patterns discussed previously, use of patterns can also decrease development time [1]. It is often discussed that even though tailoring has benefits tailoring means that the development time is increased. If the time for developing tailorable software can be decreased by using patterns it is certainly advantageous.

Table 6 Template of design pattern for use in the cooperative design process of end-user tailoring.

Design Pattern for End-user tailorable software		
Prefatory description		
• Name		
• Ranking		The authors confidence in the pattern
• Tailoring Categories		Which categories of tailoring the patterns suitable for
• Illustration		
Overall description of problem and solution		
• Problem		
• Forces	• Environment and task	Forces from environment and task that influence the choice of solution.
	• Human desires and capabilities	Forces from human desires and capabilities that have an impact on the choice of solution.
	• State of the software	Forces generated by the system state, for example software is sometimes unresponsive [14]
• General Solution		
Detailed description of solution		
• Specific Solution		Example of prior design decisions that influence the choise of solution. The forces are specific for the situation.
	• Prior design decisions	
• Diagrams		
• Consequences		
• Danger spots		
• Sample code		A short example of how to implement the pattern. Written in the language used at the company or in C++ as it is well known.
• Examples		Examples of features in applications where the pattern is used
• Related patterns		

We believe that the selection of usability patterns presented in Section 4 can act as a gateway to wider use of patterns in cooperative design projects developing end-user tailorable software. It is our hope that users as well as developers shall find the patterns beneficial and get encouraged to incorporate more patterns gradually. As the pattern are kept separate and not related in a comprehensive pattern language the patterns can be used in any type of development process, independent of other tools used in the process. It is also possible to just get inspired of what patterns to use for a specific type of tailoring and then use whatever pattern structure you prefer. But the intended use is that a team consisting of different types of stakeholders can discuss tailoring with the categorization as a base. As the categorization explicitly define both a user perspective and a system perspective it is easier to reach a consensus of what tailoring is needed. When the participants have agreed upon which type of tailoring that is needed they can continue the design process and then go further and look for what patterns should be considered for the chosen category of tailoring. The other usability scenarios that also have an architectural impact, bur not are vital to tailoring can be used as a checklist to find out if all essential usability issues are taken care of. If the participants find patterns to be useful, they can use the corresponding usability patterns for the usability scenarios they found were important for the software.

But how does our approach differ from the other approaches discussed? Borchers' approach [3] involves a pattern language that guide the team members to the next pattern. He just like us advocates patterns as a *lingua franca,* but there is a difference. When Borchers assume collaboration between the users and the usability experts and other cooperation between usability experts and developers, we advocates a direct cooperation between all the different stakeholders. We have previously not discussed usability experts at all, but we think that usability experts are closer to the software than the task and therefore we have incorporated usability experts in the term developer. The intention of having the same pattern structure for all type of patterns dealt with within the project is advocated by us as well as Borchers.

Schümmer and colleagues [17, 21] have constructed a whole process that is based on a pattern language just as Borscher. We have started in the small by introducing a small selection of vital unrelated patterns. Schümmer et.al support a iterative process and so do we. One of the advantages with patterns is that you can and are allowed to focus on one feature at a time and in an

iterative way fill up with new features and patterns. Also Schümmer et.al use patterns as means of communication and learning and their pattern structure get more detailed further down just as our.

John and Bass [2, 14] is they who have taken the most unusual approach, by explicitly name the different forces influencing the design decisions. We find their work with forces very insightful and as their findings are mirrored in our experience from industry, we felt it was essential to incorporate the forces in the pattern structure for end-user tailoring. Unlike us John and Bass have built in a sort of process in the pattern structure. For example the responsibilities of the general solution are transferred to the section of specific solutions to get a better overview of how the specific solution should look like.

The last approach, but the most well known, is the approach of the Gang of Four, Gamma et.al, [10]. Gamma et.al also have patterns that are not related in a pattern language. The main difference between Gamma's et al. approach and ours is that the patterns are mainly intended for developers and are described thereafter. But the patterns are intended for a base of communication even though within the developers' group.

7. SUMMARY

The study has resulted in a subset of usability patterns with architectural impact. The subset are matched with corresponding tailoring category to make it possible to focus on a few, vital usability patterns that is not negotiable when designing end-user tailorable software of a specific type. The selection of usability patterns is intended as a sample of how useful patterns can be in a cooperative design process. By allowing for designing with this kind of building blocks the cognitive load of the participants decreases [1] and the patterns can be a mediating artefact in the design discussions and decisions. The study also resulted in a pattern structure for patterns of end-user tailoring design. The pattern structure is a merge between several different approaches to be able to satisfy the needs of both users and developers. The patterns have to be easy to grasp and understand as well as detailed enough to be useful when implanting the software. This is achieved by starting with a prosaic description of problems and general solution and then a more detailed description of the solution is presented along with detailed diagrams and so on. This latter part aims more at the developer, but it is also our belief that interested users get more and more familiar with the pattern structure and gradually learn the meaning of, not only the beginning of the patterns, but also the more detailed and developer adjusted part.

ACKNOWLEDGMENTS

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

REFERENCES

[1] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice. Addison Wesley, Chichester, England, 1998.

[2] Bass, L. and John, B. E., "Linking Usability to Software Architecture Patterns trough Scenarios," The Journal of Systems and Software, 66(2003), 187-197.

[3] Borchers, J., A Pattern Approach to Interaction Design. John Wiley & Sons, Ltd, Chichester, England, 2001.

[4] Buschman, F., Henney, K., and Schmidt, D. C., Pattern-Oriented Software Architecture - On Patterns and Pattern Language. John Wiley & Sons Ltd, Chichester, England, 2007.

[5] Eriksson, J. and Dittrich, Y., "Combining Tailoring and Evolutionary Software Development for Rapidly Changing Business Systems," Journal of Organizational and End User Computing (JOEUC), 19, 2 (2007), 47-64.

[6] Eriksson, J., Lindeberg, O., and Dittrich, Y., "Four Categories of Tailoring as a Means of Communication," submitted to Journals of Software and Systems(2007).

[7] Ferre, X., Jusisto, N., Moreno, A. M., and Sánchez, M. I., A Software Architectural View of Usability Patterns. In the Proceedings of the INTERACT 2003, (Zürich, Switzerland, September 2003).

[8] Fischer, G., Meta-Design: Beyond User-Centered and Participatory Design. In the Proceedings of the Proceedings of HCI International 2003, (Crete, Greece, June 2003, 2003). Lawrence Erlbaum Associates, Mahwah, NJ, 88-92.

[9] Folmer, E. and Bosch, J., Usability Patterns in Software Architecture. In the Proceedings of the 10th International Conference on Human-Computer Interaction, HCII2003, (Crete, 2003.). 93-97.

[10] Gamma, E., Helm, R., Johnsson, R., and Vlissides, J., Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, Indianapolis, 1995.

[11] Gasson, S., "Human-centered vs. user-centered approaches to information system design," JITTA: Journal of Information Technology Theory and Application, 5, 2 (2003), 29-46.

[12] Henderson, A. and Kyng, M., "There's No Place Like Home: Continuing Design in Use," in Design at Work, J. GreenBaum and M. Kyng, Eds., first ed. Hillsdale, NJ: Lawrence Erlbaum, 1991, pp. 219-240.

[13] Ilvari, J. and Iivari, N., Varieties of User-Centeredness. In the Proceedings of the HICSS '06, Proceedings of the 39th Annual Hawaii International Conference on System Sciences, (Hawaii, 2006). IEEE, 176a-176a.

[14] John, B. E., Bass, L., Sanchez-Segura, M.-I., and Adams, R. J., Bringing Usability Concerns to the Design of Software Architecture. In the Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction, (Hamburg, Germany, July 11-13, 2004).

[15] Johnson, B., Woolfolk, W. W., Miller, R., and Johnson, C., Flexible Software Design - Systems Development for Changing Requirements. Auerbach Publications, Taylor & Francis Group, Boca Raton, FL, 2005.

[16] Juristo, N., Lopez, M., Moreno, A. M., and Sánchez, M. I., Improving Software Usability through Architectural Patterns. In the Proceedings of the ICSE 2003 Workshop "Bridging the Gaps Between Software Engineering and Human-Computer Interaction", (Portland, Oregon, USA, May, 2003).

[17] Lukosch, S. and Schümmer, T., "Groupware development support with technology patterns," International Journal of Human-Computer Studies, 64, 7 (2006), 599-610.

[18] Nardi, B. A., A Small Matter of Programming - Perspectives on End User Computing. MIT Press, Cambridge, 1993.

[19] Pree, W. and Sikora, H., "Design Patterns for Object-Oriented Software Development," in a Tutorial at the International Conference on Software Engineering, ICSE '97. Boston, Massachusetts, USA, 1997.

[20] Schümmer, T., Lukosch, S., and Slagter, R., Empowering End-Users: A Pattern-Centered Groupware Development Process. In the Proceedings of the 11th International Workshop, CRIWG 2005, (Porto de Galinhas, Brazil, September 25-29, 2005). Springer,

[21] Schümmer, T. and Slagter, R., The Oregon Software Development Process. In the Proceedings of the XP 2004, (Berlin/Heidelberg, 2004). Springer-Verlag, 148-156.

[22] Stevens, G., Quaisser, G., and Klann, M., "Breaking It Up: An Industrial Case Study of Component-Based Tailorable Software Design," in End-User Development, vol. 9, H. Lieberman, F. Paternò, and V. Wulf, Eds. Dordrecht, Netherlands: Springer, 2006, pp. 492.

[23] Stiemerling, O., "Component-Based Tailorability," vol. Dissertation. Bonn, Germany: Bonn University, 2000, pp. 180.

[24] Tidwell, J., Deigning Interfaces - Patterns for Effective Interaction Design. O'Reilly, 2005.

[25] Tidwell, J., "Deigning Interfaces - Patterns for Effective Interaction Design", <http://designinginterfaces.com/>, (September 13 2007).

[26] Tidwell, J., "UI Patterns and Techniques", <http://time-tripper.com/uipatterns/>, (September 13 2007).

[27] Tidwell, J., "Common Ground: A Pattern Language for Human-Computer Interface Design", <http://www.mit.edu/~jtidwell/common_ground.html>, (September 13 2007).

[28] Wesson, J. and Cowley, L., Designing with Patterns: possibilities and Pitfalls. In the Proceedings of the 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns, INTERACT 2003, (Zürich, Switzerland, September 2003, 2005). IOS Press,

An Industrial Case Study on Visualization of Dependencies between Software Measurements

Ludvig Johansson
IT University of Göteborg
412 96 Göteborg
ludvig@ituniv.se

Wilhelm Meding
Ericsson SW Research
Ericsson, Sweden
wilhelm.meding@ericsson.com

Miroslaw Staron
IT University of Göteborg
412 96 Göteborg
miroslaw@ituniv.se

ABSTRACT
Managing large software projects requires working with a large set of measurements to plan, monitor, and control the projects. The measurements can, and usually are, related to each other which raise an issue of efficiently managing the measurements by identifying, quantifying, and comparing dependencies between measurements within a project or between projects. This paper presents a case study performed at one of the units of Ericsson. The case study was designed to elicit and evaluate viable methods for visualizing dependencies between software measurements from a perspective of project and quality managers. By developing a series of prototypes, and evaluating them in interviews, we get results showing applicability of each visualization method in the context of the studied organization. The prototypes were used to visualize correlation coefficients, distribution dependencies, and project differences. The results show that even simple methods could significantly improve the work of quality managers and make the work with measurements more efficient in the organization.

Categories and Subject Descriptors
D.2.8 [**Metrics**]: *Process metrics, Product metrics.*

General Terms
Management, Measurement.

Keywords
Software metrics, visualization, quality management.

1. INTRODUCTION
Using measurements as a mean of monitoring software projects is a characteristics of mature processes and management practices. The larger the projects, the larger the data sets used and the more measurements collected. One of the daily works of quality managers is to work with measurements to assure the quality of the final product which involves identifying anomalies in data sets. Currently, the identification is based on experience and monitoring of limited number of measurements. Better use of the measurements in projects requires automated support in identifying and visualizing dependencies between measurements, especially when data sets are large. The existing visualization solutions require extensive customization work in order to be adjusted to the processes used at Ericsson and to integrate with existing toolset at the company. In this paper we identified and evaluated a set of visualization methods which do not require any initial investments nor entail large customization/integration costs.

Hence, in this paper we address the following research question:

How can dependencies between measurements be quantified and visualized in the context of a software development unit at Ericsson?

We consider both the dependencies between the measurements, and, to a limited extent, measured entities. Our research is performed in the context of software development organization, which in particular means that our work focuses on project and process measurements.

By *dependency* we mean such relationship between measurements in which a change of value of one measurement causes a change in another measurement.

The results of this study show that simple visualization techniques integrated with MS Excel and mind mapping tools could significantly improve the work of quality managers. Using mind maps to visualize dependencies between the whole (or part of) sets of variables were found to be the method which suits the evaluation criteria best, and visualizing correlation coefficients using colored MS Excel worksheets was found to be the most useful method.

The paper is structured as follows. Section 2 presents the most related research relevant for our work. Section 3 presents the context of the study – measurement systems being used at Ericsson. Section 4 describes the design of the case study and Section 5 presents the results from it. Section 6 addresses the main threats to validity of the study and Section 7 presents the conclusions.

2. RELATED WORK
In the area of software measurement and measurement dependencies, the ISO standards ISO/IEC 15939 [1] and ISO/IEC 9126-1 [2] provide a standardized way of structuring the software measurement process and preserving product quality during the process. These standards, however, are high level standards and

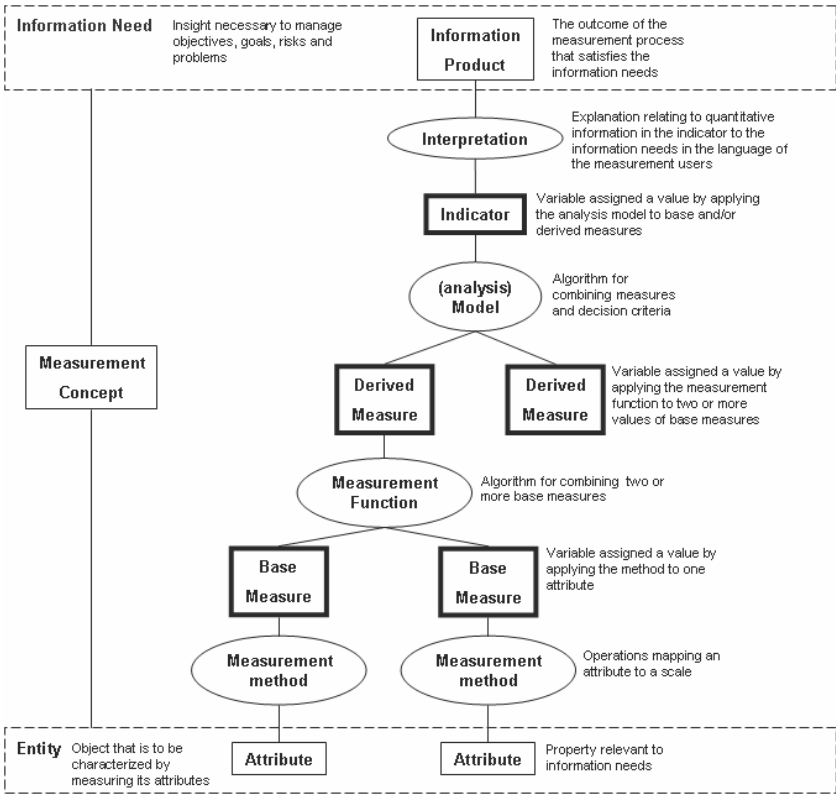


Figure 1, Software Measurement Model [1]

do not give solid theoretical background on how measurements should be used. Such fundamental application of measurement theory for software engineering is provided by Fenton and Pfleeger [3]. This study combines these two views on measurements, i.e. (a) the measurement theory perspective and (b) the ISO/IEC industrial standards perspective, in order to develop methods that are industrially applicable on theoretically solid ground.

The discussion of introducing measurement systems into an organization is, however, not covered in this study. Authors like Clark [4], Kilpi [5] Dekkers and McQuaidfor [6], Pfleeger et al. [7] and Brökcers et al. [8], describe how to/why introduce measurement systems in to an organization, and reflect on problems/solutions that measurements can result in. Their findings are used in the process of introducing the methods described in this paper into the organization.

One of the challenges in this study was to quantify measurements in a correct way – that is, whether it is possible to statistically compute dependencies: methods for statistic calculations are presented in [9, 10].

Techniques for visualizing dependencies in other areas have a wider research base than visualizing measurements in software engineering. The main focus in the existing studies of visualization is on program code dependencies, for example [11-18]. Hence, visualizing measurements dependencies is mostly about visualizing large groups of information complexes, like visualization of code dependencies and SQL dependencies, visualizing techniques provided by Spencer [19] were used as ground for identifying problems around visualizing information. Software measurements are used in the process of visualizing such aspects as code complexity, but then the focus is on the complexity and not on measurement dependency. In our research the focus is on the identification of measurement dependencies, not on complexity or size of source code.

An interesting similarity can be observed between measurements visualization and neural networks, since both include similar calculations [20]. The case-by-case comparison (described later in the paper) is based on analogy-based estimation techniques from the neural networks [21-23].

More advanced techniques for visualizing large quantities of data can be found in [24]. Although the methods presented there are applicable for our context, they required advanced visualization tools, which contradicted the requirements from the organization.

3. MEASUREMENT SYSTEMS

Ericsson is a world-wide telecommunication manufacturer. Its projects vary in size, but the majority comprises of large and long-term projects that involve both hardware and software components. Ericsson has adopted and further developed mature methods for developing software and managing projects, including managing/assuring quality of Ericsson products. The management use measurements together with expert opinions of project managers and engineers as the provider for information and a basis for making decisions – which is a common situation in mature organizations. In large software projects, however, the situation becomes hard to manage since the number of measurements used is very large, which makes it hard to manage the measurements and therefore several decisions are based on experience. In order to make the work with measurements more efficient, the studied organization at Ericsson has adopted the

ISO/IEC standard for software measurements – ISO/IEC 15939 [1].

The ISO/IEC 15939 standard defines the elements of the measurement systems as presented in Figure 1. The measurement process is driven by an *information need* (top of Figure 1). The information need is what the customer (or a stakeholder) of the measurement system wants to know, for example: 'Is the project within budget?', or 'Is the project running according to the schedule?'

In order to satisfy the information need, a series of measurements need to be examined. The measurements are collected by measuring relevant *entities*, for example, a design model, project status, or a process. An entity is a real world entity which has measurable *attributes*. The standard defines an attribute as "a property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means". The quantification of the attribute is the process of obtaining a *base measure*. Several base measurements can then be merged throughout a *measurement function* to a *derived measurement*. A measurement function is an algorithm or calculation performed to combine two or more base measures.

Further, *indicators* are created from the derived measures to provide an estimate or evaluation of specified attributes derived from the real world. It is the indicators that should fulfill the stakeholder's information need.

Table 1 presents a definition of an example measurement system which is based on a working measurement system at Ericsson.

Table 1. Defect reports measurement system - definition

Concept	Definition
Information Need	How much, compared to the budget of project X, is the cost of defect reports?
Measurable Concept	Budget deviation (budget is fixed, project cost on the other hand is dynamic)
Entity	Budget deviation
Attributes	1. Project X related defect reports 2. Cost of one defect report in project X 3. Budget of project X
Measurement Method	1. Count total number of defect reports 2. Calculate the number of hours per defect report based on data from previous projects [cost] 3. State the budget of project X (no need to calculate, it's only a number)
Base measures	1. NoD – Number of Defects 2. DC – Defect Cost 3. PB – Project Budget
Measurement Function	((NoD times DC) divided by PB) in percent
Indicator	Red/Yellow/Green
Analysis Model	Green if DM[1] < 1% Yellow if 3% > DM ≥ 1 % Red If DM ≥ 3%
Interpretation	If Red: Situation critical. Re-planning necessary. Inform steering group If Yellow: Take actions to avoid budget overrun and time plan delays If Green: No action

[1] Abbreviation for Derived Measurement

In this example the information need that the stakeholder, in this case the project manager, is concerned about is how much, compared to the project X[2] budget is the cost of defect reports.

The entity and measurable concept is the budget deviation. Attributes like project X defect reports [as a number]; Cost [in hours] of one defect report in project X and Budget [in hours] of project X are then chosen. These attributes are chosen out of experience by the developers of the measurement system (who usually have experienced as project managers). When having these attributes, a method is created to be able to measure the attribute, that is, convert the physical attribute to a numerical value to be used in mathematical calculations. The use of multiple measurements in a calculation results in obtaining derived measurements. In this example the measurement methods are: (a) count number of defect reports for X; (b) cost (in hours) of a defect report based on empirical experience and (c) budget (in hours) for project X. In this example, the indicators are set to green if the result value from the derived measurement is below 1%, yellow if between 1% and 3% and red if it is above 3%. These values are carefully selected out after a discussion with the stakeholders and based on experience from former and current projects.

An instance of this definition is presented in Table 2.

Table 2. Defect reports measurement system - instance

Concept	Definition
Values of measurements	4. NoD: 78 [defects] 5. DC: 3 [hours per defect report] 6. PB: 8000 [hours]
Derived Measurement (DM)	((58*3)/8000) * 100 % = 2,2 %
Indicator value	Yellow
Interpretation	Yellow: Project was slightly re-planned, more effort was put into solving defect before further development.

The example shows that even constructing simple indicators, one needs to be concerned with several measurements. Computing derived measures can require checking assumptions of measurements independence or dependence. The indicators are built based on these assumptions – slight deviations from established dependency relationships could make the indicators show false alarms or not indicating problematic situations. In the example above one such assumption is the cost of repairing one defect – if the cost is much lower than assumed, then this indicator would raise false alarms; if the cost is much higher than assumed, then the indicator would not inform in time about budget problems in the project.

4. CASE STUDY DESIGN

We performed our case study at Ericsson, a world-wide provider of telecom network equipment. The study was conducted at one of the quality management departments, working with measurements and measurement systems on a daily basis. The

[2] Project X can be compared to a real project at Ericsson however questions and values has been altered.

data which was used to evaluate the prototypes comes from several large software projects which the department is responsible for. The study is performed in a similar context as our previous studies (e.g. [25]).

The studied organization posed the following high-level requirements on the solutions which we should consider:

- The solutions should visualize large number of measurements
- The solutions should be able to compare dependencies between projects
- The solutions should use and/or integrate with existing toolset available at the company
- The solutions should follow the standards adopted by the company
- It should be possible to combine individual solutions into larger ones

The case study was divided into two parts – identification of viable visualization methods including elicitation of criteria for evaluating the methods, and evaluation of the visualization methods using data from historical and on-going projects at the studied unit at Ericsson. In short, our research process was:

1. Elicit criteria for comparing visualization methods – for assessing their applicability for the company.
2. Identify viable visualization methods via literature study.
3. Develop prototypes.
4. Evaluate prototypes on actual data from the company and through interviews.

The second author is working at the company and conducting both research and development in the area of software measurements. The results of the study are to be used in his work which makes this study an action research study. The third author is working closely with the company on the development of prototype measurements systems and evaluating them at the company. The first author spent the entire time of the study on site of the company.

4.1 Interviews

As the first step in the study we performed interviews with a designer of existing measurement systems, who is a quality manager with long term experience on working with measurements, project, and quality management at Ericsson. The purpose of an interview at the beginning of the study was to elicit criteria for assessing the usability of the tools. The goal of eliciting the criteria was to provide a basis for assessing the applicability of each prototype. By developing the criteria we also gained more knowledge of the non-functional requirements for each prototype. After eliciting the criteria the quality manager was asked to prioritize them using the $100 technique (in which a respondent is asked to distribute $100 for each prioritized element – larger amounts should got for the elements which are prioritized higher).

In the middle and by the end of the study we performed interviews with the same respondent, to evaluate the prototypes which were developed during this study. The criteria elicited from the interviews at the beginning were used to assess the prototypes.

The interviewer made notes during the interviews; the notes were used later during the study. All interviews were semi-structured as they contained both closed-ended, open-ended questions and the interviewee was allowed to make own remarks and comments.

4.2 Prototype development

After identifying the applicable visualization methods we created a set of prototypes to use these methods at the studied organization at Ericsson. In particular we developed a set of MS Excel add-ins using Visual Basic for Applications (VBA) that could parse the data, produce diagrams/charts, or export the data to other tools. One add-in was developed per visualization method.

We considered using dedicated visualization environments, but it was a strong requirement from the company to work with the toolset available and already adopted at the company. As a common ground, MS Excel 2003 was used in developing the prototypes as the used toolset at the company provided features to export data to MS Excel.

We used freeware mind mapping tools and hyperbolic browsers in more advanced visualization prototypes in order to test simple ways of presenting the information which MS Excel is not capable of.

The goal of developing the prototypes was to demonstrate the visualization methods and to provide our industrial partner with software to be used in their development of measurement systems.

4.3 Evaluation process

To evaluate the prototypes we used them on real data from on-going and past projects at the company. The results of running the prototypes on the data were shown to a quality manager who evaluated how the prototypes fulfilled the criteria.

The data from the ongoing project was a snapshot taken at the current time – this means that the data was not altered between evaluations of particular prototypes.

The weighted criteria are presented in Table 3.

Table 3. Evaluation Criteria

Criteria	Description	Weight
Usability (measurement systems developers)	It should be easy to use the prototypes, e.g. easy to fill in data, easy to start execution of macros, etc.	0.26
Time for execution	Execution should be performed very fast, that is, in less than a minute	0.24
Easy to overview and interpret results	It must be easy to overview and interpret results, e.g. tables, graphs, correlations, method.	0.12
Handle large sets of data	It should be possible to visualize dependencies in large data sets (e.g. more than 1000)	0.10
Comparing projects	Two different projects could be compared in the prototype showing how similar the dependencies are between the projects.	0.08
Parameters	It should be possible to use parameters to select a subset of measurements which are input to the add-in.	0.04
Maintainability	When prototype is finished, developers should be able to understand the concept and the code behind the prototype.	0.04
Magnitude of variables	If dependencies have different magnitudes (scale) it shall not affect the results	0.04
Strength of correlation	A strength of correlation should be calculated and shown in the resulting information	0.04
Usability (expert users)	The prototype could be used by other experts on measurements systems which has no prior experience in the current measurement system	0.04

During the interview the respondent was asked to assess to which degree the prototype fulfills the criteria using 5 point Likert scale: 1- totally unsatisfactory, 2- somewhat unsatisfactory, 3- neutral, 4- somewhat fulfills, 5- totally fulfills.

After the assessment we calculated the normalized score of the prototype. The normalized score was the product between the scores and the applicable criteria divided by the sum of weights of applicable criteria.

During the evaluation we recorded also qualitative comments from the respondents, including information how the prototype is supposed to contribute to the company.

5. RESULTS

By searching literature on visualization methods, we identified six viable visualization methods. Despite a significant body of research on visualization of source code, the methods were not applicable directly and required customization of the visualization tools, which in turn contradicting our requirements from the company.

In a series of interviews we evaluated the prototypes and identified their strengths and weaknesses. A summary of the interviews follows the results from the literature study.

5.1 Identified applicable visualization methods

Through literature study and the initial interview with the quality manager at Ericsson we identified two main ways of visualizing the dependencies (dependencies between measurements and dependencies between measured cases), grouped into three categories below.

5.1.1 Correlation visualization

The basic dependency between measurements is the correlation between two measurements. The correlation is an important indicator of dependency as correlated measurements should not be used when building predictive models. As the number of measurements collected in the organization was rather large, one could not be expected to manually run computations pair-wise. The rationale behind the developed prototypes was that they should support the users of measurements in their work by decreasing the time required to identify correlated variables.

As an extension to correlation visualization we also considered visualizing the results of Principal Component Analysis (PCA). PCA, however, had the disadvantage that it was hard to interpret and required visualization in more than three dimensions which was hard to obtain using available or freeware tools.

The most basic and well-known way of visualizing dependencies between two variables (measurements) is using scatter plots. If produced automatically for a set of variables, the scatter plots have an advantage that they could be used for more detailed examination of variables. An example scatter plot is presented in Figure 2.

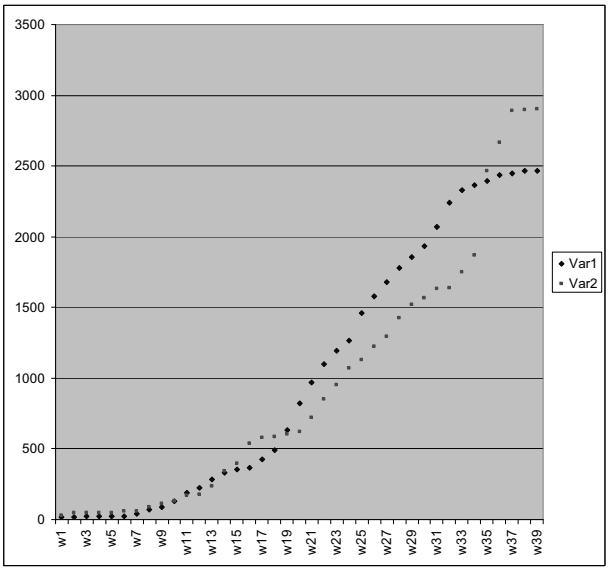


Figure 2. Visualization of dependencies using a scatter plot

The figure presents a scatter plot of two variables Var1 and Var2, which are correlated as the growing trends are observed for both variables.

Another way of visualizing the correlation between measurements is to use a matrix and a list containing colored cells with values of the correlation coefficient. An example is presented in Figure 3 as a matrix.

1		Var1	Var2	Var3	Var4
2	Var1		0,958656	0,746652	0,390364
3	Var2	0,958656		0,854176	0,287389
4	Var3	0,746652	0,854176		-0,18485
5	Var4	0,390364	0,287389	-0,18485	
6	Var5	0,989982	0,975872	0,767876	0,390388

Figure 3. Visualization of correlation coefficients – a matrix

Figure 4 shows a subset of the matrix as a list.

1	Variable 1	Variable 2	Strength
2	Var1	Var2	0,958656297
3	Var1	Var3	0,746652468
4	Var1	Var4	0,390364068

Figure 4. Visualization of correlation coefficients – a list

These prototypes are intended to provide an overview of correlations within a single data set – e.g. measurements for one project.

When building measurement systems, however, examining a single data set is sometimes insufficient. In measurement systems measurements are used based on assumptions about their dependencies, which reflect the process followed by the company. The measurements, nevertheless, tend to change over time and hence the same measurement system might provide misleading information when used at two different projects if dependencies between variables are different.. Therefore a support is needed to check whether the dependencies between measurements in two projects are indeed the same. For this purpose we created the correlation differences prototype, which visualizes the differences in correlation coefficients between two sets of measurements. An example is presented in Figure 5. Once again the colors are used to emphasize the magnitude of differences. The colors are chosen as parameters of the prototype and therefore highlight differences important for the user.

1	Pair	Difference	Sign differ
2	Var1 & Var2	0,0232875	
3	Var1 & Var4	0,0461366	
4	Var1 & Var3	0,0182627	
5	Var1 & Var5	0,0985069	
6	Var2 & Var1	0,0232875	
7	Var2 & Var4	0,0033687	
8	Var2 & Var3	0,0018969	
9	Var2 & Var5	0,1930474	
10	Var4 & Var1	0,0461366	
11	Var4 & Var2	0,0033687	

Figure 5. Visualization of differences of correlation coefficients

The result of running the prototype on two sets of data is a list of pairs of measurements and the differences between the correlation coefficients of the measurements in the pair in the two projects. The column labeled sign differ indicates whether there was a difference in the sign of the correlation coefficient (i.e. actual pair had the opposite behavior/trend in project B compared with project A?).

The difference between the correlation coefficient is to be interpreted manually based on the need. For example, when predicting quality of the project one uses regression equation which are built on one data set to predict quality using another data set. If the correlations between variables in these two data sets are significantly different, then the predictions might not be accurate. Therefore, significant differences between the correlations can be seen as an indicator of small accuracy.

Visualizing dependencies using a matrix or a list does not show a transitive dependencies – e.g. measurement A depending on B, B depending on C, etc. Therefore we developed the so-called X-Centric model prototypes using external freeware viewers: H3Viewer [26] and FreeMind [27]. This visualization method shows a network of dependent measurements, centered on a single measurement (Var1 in the example below). An example output from the FreeMind tool is presented in Figure 6. The numbers in the figure are correlation coefficients.

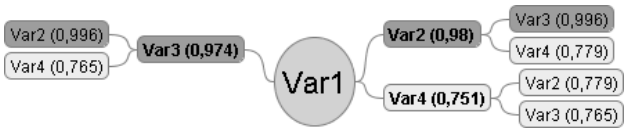


Figure 6. X-centric model visualization - FreeMind

The figure shows Var1 in the center and Var2, Var3, and Var4 which are correlated with Var1 with the strengths given in brackets – 0.98, 0.974, and 0.751 respectively. Var2 and Var4 (top left-hand corner) are correlated with Var3 with strengths given in brackets – 0.996, and 0.765 respectively.

An example visualization using the H3Viewer tool is presented in Figure 7.

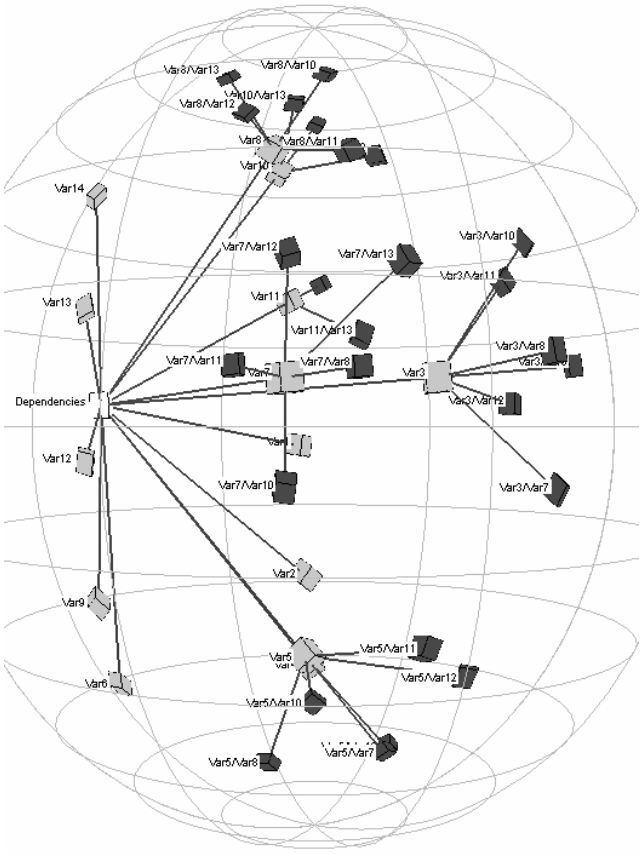


Figure 7. X-Centric model visualization – H3Viewer

Visualizing the transitive dependencies is used when building the measurement systems to identify measurements which can (if they are strongly correlated) be used interchangeably for some purposes (e.g. when building prediction models).

5.1.2 Distribution visualization

Visualizing correlations between variables shows whether the trends in the measurements are the same. The measurements, however, might be of different magnitude and/or distribution. The differences are important since the indicators in the measurement systems are built based on the values of measurements. The interpretation of indicators might depend on the distribution. Hence we developed a prototype to compare distributions. The prototype results in a bar chart with distributions of a pair of measurements and a p-value from the Chi-Square test for independence. The p-value denotes the probability that the two variables are indeed dependent.

In order to visualize the distributions between variables we used simple bar charts for graphical presentations and the chi-square test for independence to obtain the chi-square value and the probability of the measurements of being independent. An example is presented in Figure 8.

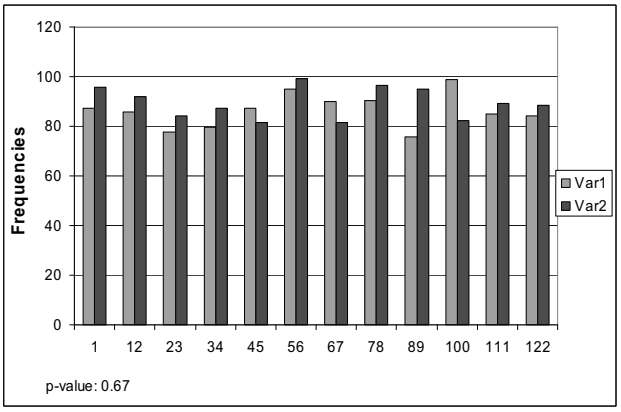


Figure 8. Visualization of distributions – bar chart showing frequencies for Var1 and Var2

The example shows that the distributions of the two variables are not different from each other, and that there is a significant probability that they are dependent on one another.

5.1.3 Case dependencies

Dependencies between the measurements provide only partial information. The information can be complemented by visualizing the dependencies between particular cases (or data subsets in the extended version of the prototype). The idea is that this comparison can identify two most similar vectors of measurements. The most similar cases to each other are believed to be dependent on each other. In this particular context we perceive this as a variation of analogy-based comparison – i.e. identifying similar cases by computing a distance between them. Analogy-based estimation has its foundation in project cost estimation [23, 28]. There, the elementary belief is that similar projects are probable to have the same behavior, for example estimated cost. In our case the rationale is that similar weeks (w.r.t. test effort) in two projects are probable to have the same characteristics (e.g. defect inflow). In analogy-based approach the estimations are derived from historical measurements. A distance function δ is calculated on l number of measurements. Weighting can be used to alter how much a measurement is supposed to affect the result. Scaling can be used if the two compared measurements are of different scale.

The distance is a weighted Euclidean distance δ, calculated using the formula:

$$\delta(p, p') = \sqrt{\sum_{i=1}^{l} w_i s_i^2 (d_i - d_i')^2}$$

Equation 1, Distance calculation for Analogy-based Estimation [21]

In Equation 1, δ stands for distance, p for points, w for weight, s for scale, d for value of a variable, while i is the index over measurement values for the data point.

The results of case dependencies is radar plot showing the most similar cases and the distance between them – an example is presented in Figure 9.

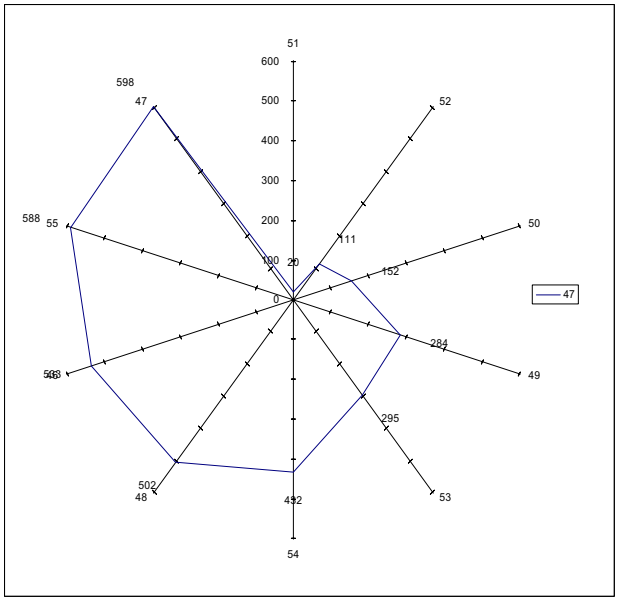


Figure 9. Visualization of case comparison – a radar plot

Figure 9 shows distances of 10 most similar cases to case 47. Each axis shows the distance between pairs of cases: case 47 and the case which is used as the name of the axis. In this example, the most similar case is case 51, as its distance to case 47 is shortest. The number of cases shown in the plot is an arbitrary number, which is a parameter in the prototype.

As an extension of comparing a single data point, the developed prototype provided a possibility to compare a series of data points and identify the most similar series in a reference data. Each data point from the series was then visualized separately using the radar plot. The similarity between the series was visualized using a colored list, as presented in Figure 10. The result is *δ/d*100%* using the symbols from Equation 1.

	Case Cmp	Case Original	Result
2	w15	w12	89%
3	w16	w13	18%
4	w17	w14	27%
5	w18	w15	22%
6	w19	w16	63%
7	w20	w17	69%
8			
9	w15	w14	81%
10	w16	w15	29%
11	w17	w16	92%
12	w18	w17	76%
13	w19	w18	95%
14	w20	w19	89%
15			
16	w15	w15	100%
17	w16	w16	78%
18	w17	w17	120%
19	w18	w18	88%
20	w19	w19	107%
21	w20	w20	100%

Figure 10. Visualization of similarities between series of data points

The example shows three series of data points (column Case Original) similar to the given series (column Case Cmp) and the differences as percentages (Result).

5.2 Evaluation of the methods

The evaluation of the methods is presented in two parts – the evaluation against the criteria and qualitative evaluation (including how the method is supposed to be used in the company).

5.2.1 Evaluation against criteria

The evaluation against the criteria is presented in Table 4. The evaluation was conducted by the quality manager. The visualization that was chosen as the best one is the X-centric view using mind-maps, although its maintainability was very low. The reason for the low value is the fact that the creation of mind-maps using the available tools could not be automated and required manual intervention every time new data points are added to the data set.

5.2.2 Qualitative evaluation

The qualitative evaluation is a summary of respondent comments recorded during the interviews when the quality manager evaluated the prototypes.

The scatter plot prototype could be used directly and for example be used at project meetings to show how measurements depend on one another. As an overview it could also be used to compare different projects which would be of use for project managers that test various changes to see how these would affect the dependencies. Today, such comparisons are not done, as the manual creation of so many plots is very time consuming.

When using the scatter plots on the real data at the company, the resulting scatter plots had one large disadvantage, namely the magnitude of the values. If two measurements had values in different scales, the scatter plot could result in that only one variable could be seen and the other variable would not be visible (due to the scaling of the plot itself). Despite this, if a basic knowledge around the dependencies exists among the stakeholders, the magnitude problem can be overseen and/or examined through the other prototypes, making this prototype a good starting point for identifying correlated measurements. .

The problem of different magnitudes of measurements in scatter plots is solved by using the correlation prototype. In the prototype another method for showing correlations was used, the Pearson's Product correlation coefficient. Using this coefficient the trends of the curves were compared while the magnitude was not crucial. Because of this, the prototype was easier to follow and interpret.

In MS Excel a list with results could easily be sorted given different criteria, which was a big benefit for the respondent. It allows easier searches in the data or shows only a subset of dependencies.

In the matrix result, a full overview of all dependencies could be seen. This gave the possibility to spot if some dependencies were

Table 4. Evaluation against the criteria (the highest score in boldface)

Criteria	Scatter plots	Correlation	X-centric (mind-map)	Correlation compare	Distribution	Analogy
Usability (developers)	5	5	5	5	5	5
Time for execution	5	5	5	5	5	5
Easy to overview and interpret results	5	4	5	4	2	4
Handle large sets of data	5	5	5	5	5	5
Comparing projects	N/A[1]	N/A	N/A	5	5	5
Parameters	5	N/A	5	N/A	5	5
Maintainability	3	2	2	3	2	3
Magnitude of variables	2	5	5	5	2	3
Strength of correlation	N/A	5	5	5	4	5
Usability (experts)	5	5	5	5	2	4
Normalized score	4.77	4.73	**4.87**	4.79	4.24	4.68

of exceptionally high or low correlation by examining the overview.

This prototype has the potential to improve the measurement systems being currently built at the studied organization at Ericsson.

H3Viewer was at an early stage rejected as a solution for modeling dependencies because of its low configurability. Strengths and correct colors for the dependencies could not be included. A hyperbolic browser was created and dependencies could be visualized, but due to the above limitations we did not include it in the evaluation.

FreeMind on the other hand, which used XML syntax with full configurability through the input file, was of good help. The clear overview with colors and correlation strengths gave a good overview of the network of dependencies. This could be used to easily and understandable show the dependency tree on how measurements were related.

One drawback of FreeMind was when more than two levels of dependencies were visualized. The resulting image spanned over a large area which was hard to get overview of when using computer screen.

Like the scatter plots, this prototype can be used to show an overview for the surrounding stakeholders during presentations. It is not certain, however, that the result will be used in the company to the same extent as the Pearson correlation.

When having a new project and a new measurement system is to be built upon the assumptions on older projects, this prototype could be used to see if the dependencies are the same in the two projects. The task of comparing projects is almost an impossible task to do by hand.

For project managers the prototype and the method could be used to track changes in the project progress/behavior compared to past experiences. When a change is introduced, a new project could be compared to older projects to see if the changes had any affect on the measurements. In this way the experts get a support in answering the question if the measurements measure the same things in the same way in the new project as in the old projects.

This prototype will, as Pearson correlation, also be useful for the company. It will be integrated in the core of the measurement systems. This prototype makes it possible for comparison of large sets of data and gives an accurate result. Today, to do this kind of comparison by hand is not possible due to the time it would take.

The comparison of the distribution of values shows how distributions of two variables could be related to each other as they have similar distributions.

This method uses the Chi-Square test for independence to obtain the p-value. During the evaluations the Chi-Square was shown not to work perfectly on the real data sets since the distributions differ too much to be compared with the Chi-Square, at least to give a meaningful result. The implementation of Chi-Square has also a limitation that it can't be computed if zero exists in the expected range. This affected the frequency table to be altered accordingly to the excepted range of values since it had to be re-configured in a way that all frequencies had at least one value. This altered frequency table gave some kind of a manipulated result which was not sufficiently good.

The magnitude of variables was also a problem. Large projects could not be compared to smaller projects since this would affect the outcome of the distribution table. In this case the measurements need to be standardized first. The magnitude of differences, however, was found to be important for the company.

Since the frequency table had to be altered to avoid the division by zero, the result could not be relied on and was difficult to interpret; hence the prototype will not be used.

The analogy-based comparison prototype had features for scaling projects to avoid magnitude problems which were found to be useful. It will be used to find matching groups of weeks in different projects to identify the most similar weeks. One drawback with the prototype is that it could be hard to overview when comparing a large number of weeks.

A particularly useful feature was the comparison between series of cases, which could help the experts to identify a series of similar data points (e.g. weeks close to finishing the project) and the similarity between them.

The prototype will be used by the developers and the analysts of the measurements systems. It will be used to compare groups of weeks to adjust the measurement systems, if needed, and could also be used to find similarities in projects. As the Pearson's correlation coefficient, this method will also be a useful for the company.

6. Validity evaluation

As every empirical research, our case study exposes some threats to validity. The validity evaluation follows the framework presented in [29].

The main *external validity* threat of the results is that this case study was performed at a single company, at one of its organizations. Even though the company cannot be regarded representative for all software industry, the context of this study is general. The evaluation criteria, however, have not been generalized to other organizations than the studied one. We are currently collecting more data from the use of measurement systems in order to increase the external validity of these results.

The *internal validity* threat, which seems to be the most influential, is the fact that the study was performed on a "static" data set – i.e. a snapshot of the data at a current time in the study. This was dictated by the time frame of the study. We intend to further evaluate the prototypes after they are integrated with the measurement systems developed at the company.

The main *construct validity* threat is that we developed the evaluation framework as part of this study. This might bias the results as there is a danger that the framework is not complete. In order to minimize this threat we took two measures: (a) developing the framework before developing the prototypes, and (b) recording the interview data to identify additional evaluation criteria (which did not happen).

7. CONCLUSIONS

Working with large number of measurements is a characteristic of large and mature organizations. As the maturity of the organizations increases the organizations seek improvements in their processes, optimizations, and better control. This leads to using more sophisticated methods for working with data being collected. In this study we evaluated several basic methods for identifying, quantifying, and visualizing dependencies between

measurements. The identified methods were evaluated empirically on data from large software projects and through a series of interviews with the quality manager working with measurements.

During the study we identified a set of criteria used to evaluate the methods. The criteria reflect the main requirements from the organization on the toolset used to work with measurements.

The results show that these simple methods are indeed very useful in working with large number of measurements as they allow identifying dependencies very efficiently. Using the evaluation criteria resulted in identifying mind maps as the best visualization method. Qualitative analysis showed that the expert found visualization of correlations between large data sets to be useful method in his work.

Our further work is focused on integrating the presented prototypes into measurement systems used at the studied Ericsson's organization.

ACKNOWLEDGMENTS

The authors would like to thank Ericsson Software Research and Software Architecture Quality Center for their support in the study. We would also like to thank managers at Ericsson who made this work possible and supported us – thank you!

REFERENCES

1. International Standard Organization and International Electrotechnical Commission, Software engineering – Software measurement process, ISO/IEC, Editor. 2002, ISO/IEC: Geneva.

2. International Standard Organization and I.E. Commission, Software engineering – Product quality Part: 1 Quality model, ISO/IEC, Editor. 2001: Geneva.

3. Fenton, N.E. and S.L. Pfleeger, Software metrics : a rigorous and practical approach. 2nd ed. 1996, London: International Thomson Computer Press. XII, 638 s.

4. Clark, B., Eight Secrets of Software Measurement, in IEEE Software. 2002. p. 12-14.

5. Kilpi, T., Implementing a Software Metrics Program at Nokia, in IEEE Software. 2001. p. 72-77.

6. Dekkers, C.A. and P.A. McQuaid, The Dangers of Using Software Metrics to (Mis) Manage in IT Professional. 2002. p. 24-30.

7. Pfleeger, S.L., et al., Status Report on Software Measurement, in IEEE Software. 1997. p. 33-34.

8. Brökcers, A., C. Differding, and G. Threin. The Role of Software Process Modelling in Planning Industrial Measurement Programs. in METRICS. 1996: IEEE.

9. Walpole, R.E., Probability & statistics for engineers & scientists. 7th ed. 2002, Upper Saddle River, NJ: Prentice Hall. xvi, 730 p.

10. Anderson, T.W., An introduction to multivariate statistical analysis. 3rd ed. Wiley series in probability and statistics. 2003, Hoboken, N.J.: Wiley-Interscience. xx, 721 p.

11. Alfert, K., F. Engelen, and A. Fronk, Experiences in three-dimensional visualization of java class relations. SDPS Journal of Design & Process Science, 2001. 5(3): p. 91-106.

12. Alfert, K. and A. Fronk. Manipulation of three-dimensional visualization of java class relations. in The Sixth World Conference on Integrated Design & Process Technology,. 2002.

13. Hendrix, D., J.H.C. II, and S. Maghsoodloo, The effectiveness of control structure diagrams in source code comprehension activities. IEEE Transactions on Software Engineering, 2002. 28(5): p. 463-477.

14. Voinea, L. and A. Telea, Visual data mining and analysis of software repositories. Computers & Graphics, 2007. 31(3): p. 410-428.

15. Kuhn, A., S. Ducasse, and T. Girba, Semantic clustering: Identifying topics in source code. Information and Software Technology, 2007. 49(3): p. 230-243.

16. Umphress, D.A., et al., Software visualizations for improving and measuring the comprehensibility of source code. Science of Computer Programming, 2006. 60(2): p. 121-133.

17. Noser, H. and P. Stucki. Dynamic 3D visualization of database-defined tree structures on the WWW by using rewriting systems. in Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop on. 2000.

18. Hing-Yan, L., et al. A multi-dimensional data visualization tool for knowledge discovery in databases. in Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International. 1995.

19. Spence, R., Information visualization: design for interaction. 2nd ed. 2007, New York: Addison Wesley.

20. Abdi, H., A Neural Network Primer. Journal of Biological Systems, 1994. 2(3): p. 247-283.

21. Auer, M., et al. Implicit analogy-based cost estimation using textual use case similarities. in International Conference on Intelligent Computing and Information Systems. 2005. Cairo.

22. Bode, J., Decision support with neural networks in the management of research and development: Concepts and application to cost estimation. Information and Management, 1998. 34(1): p. 33-40.

23. Sheppard, M. and C. Schofield, Estimating software project effort using analogies. IEEE Transactions on Software Engineering, 1997. 23(12): p. 736-743.

24. Traina, C., Jr., et al., Fast indexing and visualization of metric data sets using slim-trees. Knowledge and Data Engineering, IEEE Transactions on, 2002. 14(2): p. 244-260.

25. Staron, M. and W. Meding. Short-term Defect Inflow Prediction in Large Software Project - An Initial Evaluation. in International Conference on Empirical Assessment in Software Engineering (EASE). 2007. Keele, UK: British Computer Society.

26. Munzner, T., H3Viewer. 2001, Stanford University.

27. Freemind, FreeMind - free mind mapping software. 2007, Sourceforge.

28. Huang, S.-J. and N.-H. Chiu, Optimization of analogy weights by genetic algorithm for software effort estimation. Information and Software Technology, 2006. 48(11): p. 1034-1045.

29. Wohlin, C., et al., Experimentation in Software Engineering: An Introduction. 2000, Boston MA: Kluwer Academic Publisher.

Dependability of IT Systems in Emergency Management at Swedish Municipalities

Kim Weyns, Martin Höst
Software Engineering Research Group
Department of Computer Science, Lund University
P.O. Box 118, SE-211 00 LUND, Sweden
kim.weyns, martin.host@cs.lth.se

ABSTRACT

In recent years municipalities have become more and more dependent on IT systems for their responsibilities in a crisis situation. To avoid unexpected problems with IT systems in the aftermath of a crisis it is important that these risks are identified before a crisis occurs and that measures can be taken to reduce the dependence on systems that could be unreliable. This report describes the results of two case studies exploring how Swedish municipalities incorporate IT systems in their emergency planning. Interviews were conducted with both emergency managers and IT personnel, and data from the interviews is combined with data from a large survey. The study focuses especially on how different actors within a municipality cooperate to analyse the risks of depending on IT systems in critical situations. The study shows that today there is much room for improvement, especially in the communication between IT personnel and emergency managers.

1. INTRODUCTION

Swedish municipalities have an important active role in crisis relief. To prepare for these crisis situations, each municipality employs a number of emergency managers. Their main task consists of conducting vulnerability analyses and to use the results of these analyses to improve the municipality's ability to offer crisis relief in the aftermath of all kinds of crises while at the same time keeping their most critical services operational.

In recent years municipalities have come to depend more on IT systems for all their every day workings. For communications municipalities of course depend on landline telephone networks, mobile phone networks, webservers, email servers, etc. Other important systems are used for patient administration in health care and social care, school administration or city planning. Further, a lot of critical information is no longer stored on paper, but is only available in electronic format, either locally or even on a server located far away.

In the same way, municipalities now depend on all kinds of IT systems for their responsibilities in crisis situations. Under normal conditions, an occasional unavailability of most IT systems is fully acceptable, but during crisis situations where time is a critical factor in the relief efforts, any unexpected unavailability can have disastrous consequences. Therefore it is important that IT systems are an integral part of all major vulnerability analyses conducted. These vulnerability analyses are needed to combine information about the dependability of the different IT systems with information about how critical the systems are in different situations. A high dependency is only acceptable on systems which are highly trustworthy. Less reliable systems can also be part of emergency plans, but only if alternative solutions are available, reducing the criticality of the systems.

This vulnerability analysis is not always as straightforward as it may seem. The main complicating factor is that the information necessary is spread about over many people. Conducting the vulnerability analysis is usually the task of the emergency managers, who also work with the emergency scenarios the municipality is preparing for. Detailed information about the reliability of the IT systems is often only known to the manufacturers of the systems and the IT personnel responsible for the maintenance of the systems. In many cases this can even be external service providers that provide the support. In the worst case, when no failure statistics are systematically collected and little acceptance testing was done, no dependability information is available. Further, because reliability information is often expressed in a technical way, it can be hard to incorporate into vulnerability analyses by emergency managers without advanced knowledge about software reliability.

Detailed information on how critical certain IT systems are and how they are used in different situations is usually only available to the actual users who depend on the IT systems, and who often do not think about the crisis situations that could occur. They also often base their view of the reliability of the system completely on their own limited past experience with the system.

A second difficulty with this vulnerability analysis is that IT systems can exhibit very complex failure behaviour. A system that has worked perfectly for years, during normal operating conditions, is in no way guaranteed to work correctly in special usage scenarios. Unfortunately, these special usage scenarios are exactly what might occur during crisis

situations, which are by definition very rare events. Because of the complexity of most IT systems, it can be very hard to predict which combination of environmental and usage changes can have a large negative impact on system reliability. Even if we manage to understand this relationship between a changing usage and the dependability much better, for example through a detailed study of the sensitivity of the reliability to usage changes as described in [16, 4], it would still be difficult to predict which changes in usage we could expect during certain crisis situations because also the interactions between IT systems and users can be very complex.

A third important complicating factor is that IT systems tend to change very quickly. New systems are added regularly and old systems are almost constantly being updated. At the same time IT systems are also used for more and more new functions all the time. Good risk management would require an updated vulnerability analysis with each important change in the systems or in the way they are used. Practically this is usually impossible, but much improvement in this area is possible.

In this article we present the results of two case studies at Swedish municipalities about how they include IT systems in their emergency planning and vulnerability analyses. Additionally some results from a survey conducted among the IT safety responsible at 230 governmental actors in Sweden are presented. Section 2 presents some general background information about the role of Swedish municipalities in the Swedish emergency management system. Section 3 gives an overview of some related publications. In Section 4 the research methodology used to conduct the case studies and the survey is discussed. Section 5 discusses the main finding of the study. Some threats to the validity of the study are discussed in Section 6. Finally the main conclusions of the study and a discussion of possible future work can be found in Section 7.

2. BACKGROUND

2.1 Dependability

In this paper we will use terms such as software dependability, reliability and security as they are defined in the work from Avizienis et al. [2]. This means that dependability is defined as the most general concept, encompassing the more limited concepts of reliability, availability, safety and security. Reliability is mostly concerned with how often failures occur in the system. Availability takes into account how long the system is not functioning when failures occur. Safety is concerned with the absence of failures causing catastrophic consequences for its users and the environment, while security on the other hand, describes how sensitive the system is to external threats. Dependability takes into account all these aspects and corresponds best to the intuitive notion of how much a system can safely be depended upon by its users.

2.2 Emergency Management in Sweden

Swedish Emergency management [10] is mainly based on the principle of responsibility, which means that the in emergency conditions the responsibilities for everyday matters should still be with those governmental actors that are also responsible for these matters in normal conditions. Through the principles of proximity and geographic area responsibility, emergency management is in the first place a responsibility of the local governments. Practically, this means that municipalities are the central actor in crisis planning and crisis relief. Only with major crises that affect many municipalities the regional governments are directly involved in an operative role.

For their emergency planning, Swedish municipalities receive support from the Swedish Emergency Management Agency (SEMA, or KBM in Swedish)[1] and the regional governments. The regional councils have the responsibility to coordinate the emergency management at a regional level and to systematically review the emergency plans of the municipalities in each region and to report on this to SEMA. SEMA itself assists the regions and municipalities by supporting them in emergency planning and by providing information and guidelines. Unlike in many other countries, SEMA does not have an operative role in crisis relief.

SEMA defines a crisis as in Figure 2.2. Informally it could be stated that a crisis is when a combination of events, e.g. accidents and sabotage, result in a situation that negatively affect society in a way that hinders vital society functions. Examples of crises that are included in this definition might be terror attacks, storms, tsunamis, murders etc.

2.3 Swedish Municipalities

Sweden is divided into 290 municipalities [14]. The population of Swedish municipality range from under 2.500 in Bjurholm to 770.000 in Stockholm. Their geographic area ranges from 9 km^2 for Sundbyberg to 19.447 square km^2 for Kiruna.

The municipalities are responsible for the matters directly relating to their inhabitants and their geographic area. This means that their main responsibilities include education, child and elderly care, street maintenance and emergency management. Therefore Swedish municipalities are both service providers (for social care and education) and supervisory authorities (for environmental issues for example).

There is no standard organizational structure shared by all municipalities, but some common factors can be seen in nearly all municipalities. The main regulation governing the workings of a municipality is called the 'Local Government Act' [3]. The activities of a municipality are lead by a municipal executive committee, appointed by the elected representatives. The daily work is lead by a municipal director, who reports to the municipal executive committee.

The main activities of a municipality are usually divided over a number of administrative units, each responsible for one or more of working areas of a municipality (social service, social care, city planning, environmental issues, emergency services, culture, etc.). These activities are all external services the municipality offers to the general public. To support all these external services there is a need for a number of supporting activities, also called internal services, such as economy, technical support, housing or IT.

These internal services can be centralised for the whole mu-

"Crises are events that disrupt the functioning of society or jeopardise the conditions that govern the life of the population. They include serious crises in times of peace as well as war. Such situations demand good emergency management if they are not to undermine confidence in the Government and authorities and potentially threaten the national security and democracy of Sweden."[1]

Figure 1: SEMA's definition of a crisis

nicipality or divided over the different administrative units, depending on the organisational structure of the municipality. Many municipalities have recently brought their IT personnel into one central IT unit that offers IT services to all administrative units. This allows for a more efficient use of IT resources than before when many of the administrative units had their own separate IT personnel. We further discuss the consequences of this reorganisation in section 5.1

As said before, one of the responsibilities of a municipality is emergency management. Therefore most municipalities have one or more emergency managers who are often part of the fire department. Their responsibilities usually range from making emergency plans and conducting vulnerability analyses to organizing the information flow under an actual crisis.

3. RELATED WORK

3.1 Emergency Management

In many countries emergency planning is coordinated on a national level by a federal government agency such as the United States Federal Emergency Management Agency (FEMA), Emergency Management Australia (EMA), Public Safety Canada or the Russian Ministry of Extraordinary Situations (EMERCOM). Although the exact roles of these agencies can differ from country to country, they all support local governments in their emergency management. Because emergency management is handled differently in different countries, most countries published their own vulnerability analysis methods for use at the local level. In Sweden most publications on this topic are published in cooperation with SEMA. A good overview of Swedish emergency planning at the municipal level can be found in [5, 15]. Hallin et al.[5] also describe a scenario based method called Municipal Vulnerability Analysis (MVA).

In the private sector emergency management is usually called business continuity management. An important difference between the public and the private sector in this field is that governmental actors often have an important, active role in crisis relief and need to prepare to offer special services in the aftermath of a crisis. Business continuity management is concerned with keeping the level of service at a normal level in crisis conditions, or degrading the level of service gracefully to an acceptable level. For a governmental actor on the other hand, the unavailability of its services might be fully acceptable on any normal day, but can be critical in crisis situations. This special role in crisis relief role poses completely different demands on their emergency management procedures than those used in the private sector.

3.2 IT Management

A number of international best practice frameworks and standards have been published to help organisations obtain a higher dependability of their IT services and systems, among those ITIL [12], COBIT [7] and ISO/IEC 17799 [6]. These frameworks are much more suited to be used by large corporations with very large IT resources. For small municipalities these frameworks are too large to be of any practical use.

For this reason, SEMA published the BITS (Basic Level for IT Security) handbook [9]. This short publication is meant to give Swedish authorities a practical overview of the main administrative measures that can be taken to achieve a minimum level of IT dependability. BITS is based on international standards such as ISO/IEC 17799 [6], but BITS is much more suited for small public actors. BITS is also accompanied by BITS Plus, a web based planning tool that can be used to coordinate the work with the BITS standard. The main disadvantage with using BITS for achieving a higher dependability is that it focuses mainly on security and a lot less on reliability and safety. Most of the chapters focus only on confidentiality and integrity, without discussing other than malicious threats to the dependability of the system. Secondly, BITS also focuses mostly on the technical system level which makes it easy to loose track of the organisational level and of how critical the systems actually are for the organisation in different situations. This was also remarked in the survey described in Section 4.2. Overall this makes BITS a good tool for systematic work with IT security matters, but BITS is only part of the solution needed for evaluating the dependability of IT systems during a crisis.

Internationally, more and more research is being done on special systems that can be used in crisis relief, but many of these systems are only in the development phase. The near future will almost certainly see a serious rise in the number of IT systems used in crisis situations. So far most of these systems are only considered as an extra tool in the aftermath of a crisis, but as these tools become more common, we come to depend on them more and more. Just like when people start using a mobile phone, they first see it as a tool that just makes some things a bit easier. However, after using a mobile phone for some time, they can no longer imagine how they could ever manage without a mobile phone. Therefore, extra caution is warranted when these crisis relief systems are ready to be used in real emergency situations. To be able to use these systems efficiently, and to be able to evaluate the dependency on these systems, it is even more important to fully integrate these IT systems into emergency management and include them in the vulnerability analyses that are conducted.

4. RESEARCH METHODOLOGY

The research in this report combines results from three different sources: an elaborate literature study, data collected from case studies at two Swedish municipalities and the data of a survey conducted by SEMA among all Swedish munic-

ipalities and a series of other governmental actors. The two case studies are described in Section 4.1 and the survey is elaborated upon in Section 4.2.

4.1 Case Studies

The main part of this research was conducted in two case studies at two different Swedish municipalities. The municipalities were specially selected because they had shown an interest in the topic of IT systems in emergency management in previous contacts with SEMA or with other members of our research project.

Municipality A is a large Swedish municipality consisting of a major Swedish harbour city and the surrounding urban areas. Municipality B on the other hand is a small municipality consisting of two suburbs of a large Swedish city. The two municipalities are substantially different in many important ways. Municipality A has roughly 6 times more inhabitants and also employs about 7 times more people. Also from a vulnerability perspective there are large differences. Municipality A houses a lot of industry and is an important national hub for the transport of dangerous goods. During the last years the municipality has gone through some major emergency situations of different types. Municipality B on the other hand has a much lower risk profile and has not experienced any major emergency situations in the last 15 years.

To understand how these municipalities assess the dependability of their IT systems in emergency situations, a series of interviews were conducted with emergency managers and IT personnel at both municipalities. Further a number of documents concerning IT strategies, organisational structures and vulnerability analysis were also collected and studied, both before and after the interviews. Because the nature of the research was strongly explorative, each consecutive interview or document was studied immediately and this information was used to improve the preparations for the next interviews. The disadvantage with this method is that it might introduce a bias in the next interviews, but it was considered that the advantage of a more informed preparation for further interviews outweighed this disadvantage, especially since some of this researcher bias is inevitable when all interviews are conducted by the same researchers.

The interviews were conducted as open interviews [13], with a lot of freedom for the respondents to give their view on the issues at hand. All the interviews used the same basic list of open questions, but they were only used to make sure the interviews covered all the necessary topics, not to decide the order of the topics. Because different municipalities have such different ways of working, it was not possible to compile a list of very specific questions that could be used for all the interviews. Often it was necessary for the interviewees to first explain a number of other aspects before they could completely answer a certain question. The advantage of this freedom in the interview is that the respondents had the freedom to stress the parts they see as the most important. The main disadvantage is that the analysis of the interviews becomes harder because there is not standard structure.

For the first municipality, interviews were conducted with two emergency managers and one former IT manager, currently working at the social care department as project manager, specialised in IT projects. At municipality B, interviews were conducted with one emergency manager responsible for IT safety and one IT technician. At both municipalities the emergency managers were interviewed first, since they are easier to contact for an outsider. They were then asked to provide contact information to suitable contacts in the IT department.

For the analysis, all interviews except the first one were recorded and transcribed in full. During the transcription they were also translated from Swedish to English to facilitate the analysis. As recommended by Robson [13], a number of coding categories were used to reduce the amount of data to be studied. For the coding two independent researchers went through all the transcribed text and coded all passage that related to one or more of the following categories and subcategories:

- Organisation
  - Organisational structure
  - Responsibility for the IT systems
  - Organisational changes
- Risk analysis
  - Risk analysis activities
  - Identification of critical systems
  - Prioritisation of IT support
- Communication with the IT personnel
  - Ways of communication
  - Driving force for communication
- Service level agreements (SLA)
  - SLA form
  - SLA content
- Practical examples
  - Past problems
  - Frustrations
  - Implemented solutions and practices.

These final categories are the result of a stepwise improvement from an initial set of categories based on the main concepts in the research. The coding helps us to identify statements that logically belong together but are spread out over the text. The coding was not a goal on its own, but an analysis tool and therefore the categories were not defined too strictly beforehand and it was left to the researchers doing the coding to fine tune the categories.

The first category collects statements about the personnel involved in evaluating the dependability of the IT systems. The focus of this category is on how the responsibilities are divided between the different people involved. The second category contains all data about how vulnerability analysis

is performed at the municipalities under study, with special focus on how the IT systems are analysed. The third category collects the information about how and when the IT unit communicates with the rest of the municipality's personnel. The fourth top category is about service level agreements in the very broad sense, so everything about the level of services expected from the IT systems at the municipalities, and how this is specified or agreed upon. The last category collects all the practical examples that were discussed during the interviews that were most illustrative for the issues discussed in this report.

After two separate researchers, i.e. the authors of this paper, marked the interviews according to this categorisation, their lists were merged and the excerpts in every category and subcategory were analysed. Since the interviews often returned to the same topic, and because different people in the same organisation were interviewed, a triangulation can be done to check the consistency of the interviewees' answers.

For the analysis both within and across the two municipalities the technique of explanation building as described by Yin [17] was used. Special attention was given to those issues where the respondents disagreed or gave conflicting answers. More details on how the conclusions were reached from the data can be found in the discussion of the study's findings in Section 5.

4.2 Survey by SEMA

In May 2005, SEMA conducted a survey among 368 IT security managers at Swedish municipalities, regional governments and different public authorities. A first analysis of the 230 answers to the survey they received was published shortly afterwards [8].

The survey consisted of between 14 and 30 questions, depending on the chosen alternatives. The majority of the questions were multiple choice questions where respondents were asked to rate something on a scale from 1 to 5. A substantial part of the questions were open questions that gave the respondents the chance to explain their answers in more detail.

The goal of the survey was to assess the capabilities of different governmental actors in the field of IT security. Within IT security the survey focused mostly on the methods and standards used and how SEMA's support towards the governmental actors could be improved. The respondents were also asked to make an assessment of the maturity of their organisation and different members of their organisation in IT security.

For this report we had access to all the raw data from the survey, but all names were removed for integrity reasons. It was still possible to determine if a series of answers came from a municipality or a regional government, but answers could no longer be connected to a specific public actor. The answers to the multiple choice questions were mostly used to see how common the use of the different methods and standards is. The answers to the open questions were analysed in a similar way as the interviews in the case studies. The main conclusion and a graphical analysis of the multiple choice questions can be found in the survey report [8].

When considering the validity of the data collected from the survey we need to keep in mind that the answers only reflect the view of the IT security responsible at each public authority. To get a complete picture other roles in the municipality should be included in the survey too. Secondly, it might be possible that those public actors that have the lowest level of maturity in IT security did not care to answer the survey, and this would make that the results can not be generalized blindly. Further it is important to see that the focus of the survey and the case studies is slightly different. The survey focussed on security, while the case studies were concerned with dependability.

5. FINDINGS

This section contains the main findings from the case studies and the survey. Each of the next sections discusses the conclusions that can be drawn from the excerpts that were coded in to the corresponding categories and subcategories. Therefore the following sections follow roughly the structure of the categories listed in Section 4.1, though the order has been changed to facilitate the discourse.

5.1 Organisation of IT Services

5.1.1 System Responsibility

In both municipalities that participated in the study there is a central IT unit responsible for the maintenance of the IT systems. For some systems, the maintenance is done with the help of suppliers or external consultants. When it comes to the final responsibility for the system, both municipalities make a distinction between those systems that are common for the whole municipality and those that are specific for one department. The former systems such as the email system, the network or the operating systems are the direct responsibility of the IT department. The latter systems such as the economy system or systems used in social care are the responsibility of the specific departments. This responsibility means they decide about the acquisition, the updates and the evaluation of the systems. The maintenance for both types of systems can be performed by either the IT department or by external consultants, for example from the supplier of the system. The contracts with the supplier can be signed with or without some involvement of the IT department.

The main advantage of this approach is that the main responsibility for all the systems lies with those who have the most knowledge of the application area of the system. This approach also has a number of problems, especially in the cooperation between the IT department and the people responsible for the systems owned by the different departments. Because they are in different departments with different goals, there is often a conflict relationship between them prohibiting a good cooperation and exchange of necessary information.

A first problem lies in the evaluation of the dependability of the systems. Since the IT department is responsible for the maintenance they are contacted in case of any problems, but it is not their responsibility to collect failure statistics, as expressed in Quote 1. The IT personnel has the ungrateful role of having to maintain these systems while they can

not directly influence their administration. The responsible of the system on the other hand, is then not even notified of all the problems, and can not get a full picture of the dependability of the system. In municipality A, the IT department has a help desk that coordinates the maintenance work of the IT department. In municipality B, users contact one of the employees of the IT department directly on their mobile phone, making it even harder to collect failure statistics. Further, concerning the service that is outsourced to external suppliers, some failures are reported directly to the supplier, while others are reported to the supplier through the IT department.

Quote 1
We don't do any organised collection of statistics now, we just try to solve the problems that pop up. – IT Technician, Municipality B

A second problem is that most of the systems owned by the various departments are dependent on the operating systems and the network administrated by the IT department. Since both groups have the individual responsibility to decide about major updates to their systems, this can create problems when these are not communicated well in advance.

A third problem is that in this organisational structure the IT-departments do not have any own technical personnel that can advice them on the technical details that are involved in the administration of the systems they are responsible for. To be able to take full responsibility for the systems not only a good understanding of the purpose of the systems but also a good technical understanding of the workings of the system is necessary. This can lead to responsibilities implicitly being shifted to the IT department where they do not belong, just because the different departments do not immediately know how to deal with them. This is for example complained about in the survey as can be seen in Quote 2.

Quote 2
To define the limits of their area of responsibility to make sure that the responsibility is where it should be. This is necessary to avoid that the stress lies on the technology in stead of the processes. We are not good enough at explaining that there are some parts where the different departments must take responsibility. Today it is automatically the IT unit that must take responsibility for IT matters for which no-one else takes responsibility. This is not good. – Survey answer to the question: What do you think the IT personnel could get better at concerning IT dependability?

Another common problem with the organisation of the IT department is that in old organisational structures IT is still considered to be a part of the economy department and the Chief Information Officer (CIO), or a similar function, still reports to the head of the economy department, and not to the director of the municipality directly. This is also referred to as an important inhibitor to a good cooperation between the IT department and the rest of an organisation in the IT governance literature, for example in the work by Luftman [11]. This problem was also visible at municipality B and was complained about in the survey.

In municipality B, IT safety is a responsibility of one of the emergency managers at the municipality. The advantage of this role is that he can lift these safety issues immediately to the highest levels in the municipality where the legal responsibility for all safety matters in the municipality lies. On the other hand, the danger is that the IT department feels relieved of all safety responsibilities although their expertise is indispensable for evaluating this safety.

5.1.2 Internal Communication

An often recurring complaint, in the case studies and the survey, is a lack of real understanding between the IT department and the users. Users complain that the IT personnel does not understand what they expect of their systems, as for example in Quote 3. The IT personnel on the other hand complains that the users do not understand the risks involved with IT systems, especially concerning security.

Quote 3
We have generators and we can provide backup power to our IT systems very long. Quality of the IT systems is harder. We have discussed this a lot. Also with our IT technicians, but they focus often on the wrong things. – Emergency Manager, Municipality B

This lack of understanding is a consequence of the communication problems between both parties. Both municipalities under study lacked a forum where the IT department and the users could discuss important IT issues together, as discussed in Quote 4. For some major discussions working groups are created that include representatives of suppliers, users and the IT department, but this is done too seldom. In the worst case the only communication occurs when a failure of a software system occurs and the IT department has to be notified to fix the problem.

Quote 4
They always want to buy a new server for every application, but that is not always necessary. But it is not us who decides, we just hopefully get asked, though often too late. – IT Technician, Municipality B

For example, when it comes to communicating about major updates to the systems, under the responsibility of either the IT department or the other departments, both municipalities admitted that they had encountered problems in the past. All people involved knew that the best way would be to discuss any major updates with all parties involved before the decision to update is made final, but in practice many decision where made unilaterally and sometimes the other parties were not even notified in advance of the update.

Another common complaint about the communication between users and the IT department is that the communication from the IT department is too technical. Outside the IT department there is not enough technical knowledge to understand the technical details of the system, while the IT department does not manage to communicate their message without resorting to technical details. This adds to the frustration of parties, and results in the IT department not being consulted as often as necessary for important decisions.

5.1.3 Service Level Agreements

Both municipalities in the study have some service level agreements (SLAs) with their external suppliers but have no service level agreements at all with their own IT department. In a small municipality it would probably be too tedious to write formal agreements that should be considered as binding contracts. Nevertheless, some written communication where users and the IT department discuss the level of service, could bring clear advantages to both parties. For example in municipality B, the IT department tries to always have some IT personnel reachable to provide service, even in weekends and at night in case there is a need for urgent IT support for critical systems. This level of service is in no way guaranteed to the rest of the municipality, but is just done because the IT department considers it reasonable.

Without SLAs the IT department is expected to deliver services at best effort, but without any specifications what level this is. In this situation, all failures are considered as faults of the IT department to deliver satisfactory service. Further there are no written agreement as to which systems should have a high availability, and the IT personnel estimates from experience which systems are most critical to prioritise their work. Service level agreements would give the IT department a stronger position when asking for resources to deliver a necessary level of service and at the same time protect them from user expecting an impossibly high level of service, as expressed in Quote 5.

Quote 5
There are 1000 reasons for having a service level agreement, but the one reason for not having it is that without one, the IT department is not obligated to anything. They do not see that it could also be a defence for them that they can not be blamed for not delivering something they before clearly stated they could not deliver. – Project Manager, Municipality A

The advantage of SLAs for the users is that they know what to expect, and what not to expect, from their IT systems. This way they can avoid both depending on unreliable systems and investing in unnecessary backup solutions for sufficiently reliable systems. This problem is expressed in Quote 6 from a project manager at municipality A.

Quote 6
If the IT department can explicitly state that they can not give us any guarantees, we have good reason to invest some extra millions on this side to secure our systems. But now we have no arguments to justify this cost here. – Project Manager, Municipality A

Even the service level agreements with external suppliers are often not well planned and not adapted to the level of quality actually demanded by the users of the systems. For example at municipality B, the maintenance contract with their supplier of routers guaranteed on-site service within 8 hours. This number was agreed upon many years ago, and nobody seems to know exactly why it once was set at 8 hours. The importance of the internal network for the daily operations at the municipality has definitely increased drastically since this decision was taken. This example shows there are no routines in place to regularly re-evaluate important service level agreements.

Service level agreements are closely connected to measurements. The writing of service level agreements forces an organisation to think about how the quality of its IT systems can be measured. Just as both municipalities lack service level agreement for most of their systems, they also lack the possibility to measure the quality of their IT systems. Access to such measurements would give them a possibility to concentrate their resources better to improve the weakest links in their critical systems.

5.2 Emergency Management

Every municipality has a number of emergency managers responsible for preparing the municipality for possible crisis situations. An important part of this task is to help all the departments in the municipality to conduct risk and vulnerability analyses and to produce emergency plans. The risk analyses can only be conducted by the personnel of each department, because they are the only ones that have the necessary knowledge about how emergency situations could influence their work. The emergency managers help them in this task by instructing them in the methods that can be used, and reminding them to keep their emergency plans updated.

The most commonly used methods for risk and vulnerability analysis are scenario-based, as for example the method developed by Hallin et al. [5]. Most municipalities also organise regular, scenario-based emergency exercises to test their emergency planning. The emergency management of a municipality often results in a number of simple measures that can be taken to seriously reduce the probability or effect of possible crisis situations.

The emergency managers are also responsible for planning the specific responsibilities of the municipalities in crisis relief and information spreading during a crisis. All municipalities are required to have a crisis central that can be used to coordinate the relief effort during and in the immediate aftermath of a crisis. SEMA also assists municipalities in setting up such a crisis central and analysing which facilities are required. IT systems, and especially communication systems, are an important part of the equipment available

in a crisis central.

Although IT systems can play an important role in the aftermath of a crisis, they are seldom included in the emergency plans and risk analyses that are conducted. Emergency managers would like to include these systems, but in practice they do not manage to cooperate with the IT department to do so. In municipality A, the emergency management of the social care department is planned to be completely independent of IT systems. This means, for example, that all critical information is printed out on a very regular basis and communication plans are ready that do not rely on modern technology. As the project manager explained, this is a safe solution, since it means they are prepared for a complete failure of all IT systems, but it is also a serious overhead cost that is only necessary because they do not manage to analyse the risks of depending on their IT systems. If they would manage to include the IT systems in the risk analyses, they would be able to evaluate which systems are reliable enough to depend upon in different emergency situations, and they could safely reduce this overhead cost. Because the IT systems are not part of the emergency plans, they can also not be used as efficiently in a crisis if they turn out to be reliable after all.

In municipality B, a crisis central was installed with the help of SEMA and a number of external consultants. Although this room contains a number of computers and network connections, the IT department was not involved in the development of this room. The systems in this room are meant to be used in crisis situations and have redundant phone and internet connections. The IT department also maintains the systems in this room, but they have no responsibility for the reliability of these systems and are not involved in any strategic planning of how the systems in this room should be updated or replaced.

When the IT department is not involved in emergency planning, as expressed in Quotes 7 and 8, they are also not aware of which systems are critical during different crisis situations and they can not correctly prioritise their maintenance work without receiving specific instructions during a crisis. This also means that IT systems are seldom involved in emergency exercises. Useful lessons could be learned from exercises such as regularly trying to restore a system from backup, or measuring the behaviour of the network when one or more routers are disabled. When this kind of statistics is available it can be taken into account in the emergency planning.

Quote 7
We are not involved in making emergency plans. It's not something we think about. – IT Technician, Municipality B

Quote 8
I don't know what the rules are for prioritised service in an emergency. Nobody said that one computer is more important than the others. – IT Technician, Municipality B

5.3 Common Problems

In this section we summarize the main problems the studied municipalities experienced when trying to integrate their IT systems in their emergency planning.

A first recurring problem is the lack of good supporting tools or standards. BITS [9], the brochure with guidelines published by SEMA, is used by 75% of the municipalities that answered the survey, but BITS Plus, the tool that was added more recently, was used by only 28%. BITS is more focused on security than reliability, and the focus is therefore more on the systems as separate units, and not on how the systems fit in to the overall activities of the municipality, as also remarked in the survey as in Quote 9. For this reason, BITS is not ideal for a complete dependability analysis, and might even lead to some aspects being forgotten when it is not complemented with other risk analysis tools or methods that incorporate the IT systems. The international standards and best practice frameworks such as ITIL and COBIT discussed in Section 3.2 are too large and too much focused on companies to be very useful to most municipalities.

Quote 9
The main disadvantage of BITS is that it uses an object-oriented model for IT dependability, instead of a process-oriented model. This means it sees IT systems as isolated objects, in stead of starting from the information processes that are provided or supported by the system. – Survey answer to the question: What do you think could be improved about BITS?

For this reason, municipality B has started developing their own risk management tool, with special focus on following up the whole process from identification of possible risks to mitigation. When the system is completed, it is meant to be used by all departments in the municipality. At the time of this research, the systems was however only just being deployed and was not used for documenting IT risks yet.

A second major problem that was observed at both the municipalities was the problem with defining who is responsible for evaluating the dependability of the IT systems in crisis situations. This task requires the cooperation between the emergency managers, the IT department and the department owning the system. In practice, because of the communication problems discussed before, this can lead to this issue being overlooked when nobody takes the responsibility to organize a working group to tackle this problem. Especially if the IT department is not involved in the strategical discussions about the IT systems, they limit themselves to the daily maintenance of the systems and only perform technical long-term improvements when explicitly asked. This can for example be observed in Quote 10

Quote 10
– Interviewer: Computers have become more critical in the last years. Did you recently re-evaluate the 4 hour service agreement with your network supplier?
– No, this is something the users of the applications should worry about, not us. We only have a responsibility for the maintenance of our systems: the network, the mail servers, and file servers. – IT Technician, Municipality B

Another problem is the users' and emergency managers' limited understanding of the dependability issues of IT systems. Especially concerning security, as shown clearly in the survey, the IT department often complains about the negligence of the users. Also concerning the reliability, the users do not have enough technical knowledge to understand the IT systems. When they want to conduct a risk analysis of the IT systems they need this technical knowledge to be able to understand all the threats to the reliability of the system, their probability and possible consequences. Often it is assumed that the IT systems can be depended upon in a crisis, even if there is no evidence of their reliability.

Finally, a typical problem with IT systems is their fast evolution. New IT systems are installed every year and updates are done even more regularly. Adding new systems or new functionality to old systems changes both the reliability of the system and the dependence on the system. When the municipalities already have some risk analyses of their IT systems, they do not manage to keep these analyses updated to reflect the latest functionality of the IT systems. This is especially important since the dependence on the IT systems is increasing continuously. At first, after a new system has been installed, the system is usually only considered an extra asset that could be useful in a crisis situation, even if it not critically necessary because the old alternatives are still available. At this time the dependability of the system is not critical, but when the user get more used to having the new system around, the alternative systems are neglected and the new systems can get more and more critical. When these changes occur gradually, they are sometimes only noticed too late and systems can become critical without their dependability ever having been seriously evaluated.

6. VALIDITY DISCUSSION

A number of possible threats for the validity can be identified for this study. Concerning external validity it is important to understand that the results of the case studies can not be generalized in the same way as the results of the survey.

The majority of the 290 Swedish municipalities participated in the survey, and the results to the multiple choice questions can be therefore be considered statistically representative. The open questions in the survey were only answered by few respondents and can not be generalized in the same way.

The case studies on the other hand, studied only two municipalities, and can not so easily be generalized to all Swedish municipalities. This was of course also not the goal of the study. The goal of the explorative study was to get some understanding for the problems that municipalities are facing when trying to include their IT systems in their emergency management. Even though many of the same problems occur at both the municipalities under study, this is no proof that they appear in all Swedish municipalities. When combining the survey and the case studies, we can at least conclude that some of the problems are very common, and we can suspect that some of their causes and effects is probably the same for many more Swedish municipalities.

An important threat to the validity in this study is the possibility of researcher bias. All the interviews were conducted by the same researchers and the conclusions from the first interviews were used to steer the later ones. Because of the open form of the interviews, it would be even easier for the researchers to steer the respondents to certain conclusions. To minimise the effect of researcher bias, the interviews were conducted with two researchers present and extra care was given to let the interviewees tell their own story, without guiding their answers. In the analysis of the interviews the possibility of researcher bias was constantly taken into account when building explanations.

A threat to the construct validity that is often present when data is collected through interviews is the possibility that the participants are focusing too much on their own side of the story and give a distorted view of reality. One reason for this is that people do not have perfect recollection, and only remember a part of what happened. A second reason is that people automatically try to defend their own actions, and although they would hopefully not lie deliberately, they might neglect to tell some things that makes them look bad. Through the use of triangulation, by interviewing different people at the same municipality and by asking different questions concerning the same topic, the effect of this can be reduced. Overall, the interviewees were not afraid at all to talk about problems they were experiencing or had experienced in the past. Because all official documents that are not declared classified are automatically public in Sweden, it was also no problem to gain access to any documents requested for analysis.

A final important threat to the validity is that the municipalities that were studied are listed as good examples of emergency management on SEMA's website: municipality A for using the MVA [5] technique and assisting in the development of this technique, and municipality B for the risk incident reporting system they developed. This is an indication that both municipalities might be more mature in handling these issues than most other Swedish municipalities. For this exploratory study this was considered an advantage, since this allowed us to interview more experienced participants, but it makes the results harder to generalize.

7. CONCLUSIONS AND FUTURE WORK

In this report we studied how municipalities in Sweden evaluate the dependability of their IT systems in possible crisis situations. A first set of case studies and the results of a survey have given us a better understanding of the main challenges involved.

In the case studies we noted a number of problem areas. The main problem is that the studied municipalities lack a forum where preventive measures concerning IT dependability issues can be discussed. All involved parties do their best

in contributing to the dependability of the systems, but no cooperation to discuss these matters on a strategical level is present. Therefore, those responsibilities that lie on the border between different people's areas of responsibility are often given too little attention.

Now that we have identified a problem and possibility for improvement with how municipalities deal with dependability of their IT systems, a next logical step is to start working towards a tool that can help municipalities improve in this field. In the end this should result in a process improvement model that is simple enough to be applied even by small municipalities, but that at the same time can make a big difference. The focus of this improvement model should be in stimulating the communication about these issues between the IT personnel, the emergency managers and the users of the different IT systems in the municipality.

The first step towards this goal is to develop a measurement scale and tool that municipalities can use to assess how mature they are in handling this issue and in which areas there is most room for improvement. The next step is then to evaluate this measurement tool in practice and improve it based on this evaluation.

With the help of this measurement tool we can then start to develop a process improvement model based on these measurements that helps municipalities reach a higher level of maturity in dealing with dependability issues and to sustain these improvements. The final step would then be to evaluate this complete maturity model in a practical setting at one or more municipalities while continuously improving it based on these experiences and the feedback we receive.

Acknowledgements

The authors would specially like to thank all the participants in the interview study.

The work is partly funded by the Swedish Emergency Management Agency under grant for FRIVA, Framework Programme for Risk and Vulnerability Analysis of Technological and Social Systems.

8. REFERENCES

[1] Swedish Emergency Management Agency, `http://www.krisberedskapsmyndigheten.se/`.

[2] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. *Research Report N01145, LAAS-CNRS*, 2001.

[3] Finansdepartementet KL. Kommunallag (1991:900), 1991.

[4] K. Goševa-Popstojanova and S. Kamavaram. Software reliability estimation under uncertainty:generalization of the method of moments. *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, pages 209–218, 2004.

[5] P.-O. Hallin, J. Nilsson, and N. Olofsson. Krishantering på svenska: Teori och praktik. Technical report, 2004.

[6] International Organization for Standardization. ISO-IEC 17799: Information technology - Security techniques - Code of practice for information security management, 2005.

[7] IT Governance Institute, USA. Control objectives for information and related technologies (COBIT) (3rd ed.), 2000.

[8] K. Kalmelid and J. Gustavsson. Inventering av kompetensbehov m.m. inom informationssäkerhet i offentlig sektor. Technical report, Rapport, Informationssäkerhets- och analysenheten, Krisberedskapsmyndigheten, 2005.

[9] Krisberedskapsmyndigheten. Basic Level for IT Security (BITS). Technical report, SEMA recommends 2003:2, 2003.

[10] Krisberedskapsmyndigheten. Samhällets krisberedskap - Inriktning för verksamheten 2007. Technical report, Planeringsprocessen 2005:3, 2005.

[11] J. N. Luftman. *Managing the Information Technology Resource: Leadership in the Information Age.* Prentice-Hall, 2003.

[12] Office of Government Commerce. Information Technology Infrastructure Library (ITIL) Version 3, 2007.

[13] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-researchers (Regional Surveys of the World).* Blackwell Publishers, 2002.

[14] Swedish Association of Local Authorities and Regions. Levels of Local Democracy in Sweden. Technical report, Swedish Association of Local Authorities and Regions, 2005.

[15] Vägledning från Krisberedskapsmyndigheten. Kommunens Plan för Hantering av Extraordinära Händelser. Technical report, KBM Rekommenderar 2004:1, 2004.

[16] K. Weyns and P. Runeson. Sensitivity of Software System Reliability to Usage Profile Changes. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, pages 1440–1444, 2007.

[17] R. K. Yin. *Case Study Research: Design and Methods.* SAGE Publications Ltd, 2003.

Prerequisites for Software Cost Estimation in Automotive Development Projects

- A Case Study

Ana Magazinovic
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
+46(0)31 772 57 35
ana.magazinovic@chalmers.se

Joakim Pernstål
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
pernstal@chalmers.se

Peter Öhman
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
+46(0)31 772 36 64
peter.ohman@chalmers.se

ABSTRACT

Cost estimation is an important yet difficult part of project planning. The software industry is very familiar with cost overruns and estimation error problems. As the amount of software components is increasing in modern vehicles, so are the problems with cost estimation errors.

This paper presents the results of an explorative case study done at a Swedish automotive company in order to understand the issues that affect software cost estimation in system development projects. Eleven interviews were conducted with software professionals working at the company. Investigator triangulation was used during the data collection and analysis to assure the validity of the results. Fifteen issues were identified, five of which have not been mentioned in the literature prior to this study.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *Cost estimation, Time estimation*

General Terms

Management, Measurement, Economics, Human Factors

Keywords

Cost Estimation, Case Study, Empirical Software Engineering, Automotive

1. INTRODUCTION AND RELATED WORK

Cost estimation is an important part of the project planning process. This issue has been addressed by Lederer and Prasad [16] who found that the average importance rating among the software managers and other professionals participating in their study was highly important. Most software organizations find the cost estimation activity to be a necessary part of the project planning. According to Heemstra [7] 65% of the organizations estimate projects as a rule, and according to Lederer and Prasad [16] 87% of organization's large projects are estimated.

Just as cost estimation is an important part of project planning, it is also considered a problem. In their paper [17] Moores and Edvards report that 91% of the companies in their study see cost estimation as a problem.

The figures above are taken from studies concentrating on software industry; however, the amount of software is now increasing in embedded vehicle systems and the automotive industry is facing the same problems as the software industry. For the last 30 years the share of electronics and software has been growing exponentially in vehicles since the major part of new innovations in vehicles are realized with software and electronics. According to Grimm [5] and Broy [3] up to 40% of production costs are spent on electronic components and software in the premium cars. Today's premium cars contain approximately 70 Electronic Control Units (ECU) controlling infotainment, climate and speed and new communication networks are added continuously. Further, the new software based systems tend to become more complex which makes estimation even more difficult.

With small profit margins within the automotive industry on the one hand and cost overruns in the software industry [15] [16] on the other hand improved cost control for development of software in vehicles is needed in order to decrease cost estimation errors. The prerequisites for successful cost estimation need to be explored and understood before the right estimation method can be proposed and adjusted to the company.

From a research point of view, according to Jorgensen and Sheppard [13], over 50% of the efforts done in the area of Software Cost Estimation research is concentrated on researching the different methods used to estimate effort and cost of software development, such as Expert judgment explored in the papers by Hill and Allen [8] and Hughes [10], Algorithmic models researched by Boehm and Clark in [2] and Jongmoon and Boehm in [1] and Estimation by analogy explored in the Walkerden paper [21]. Much of the work has been spent on developing own

methods and on history based evolution. However, there are exceptions that point at the need of research concerning organizational issues such as problems created by management and politics. In the Laderer and Prasad paper [15] one of the conclusions is that political and management control issues should be given more attention in cost estimation research, and in her book [14] Kitchenham says that before improving the estimation process, there is a need to explore whether the problem is really an estimation problem and not a management problem.

Thus, the purpose of this paper is to understand the factors that affect the prerequisites for successful software cost estimation in system development projects in automotive domain. The objective of the study that the paper is based on is exploring the underlying factors that affect the software cost estimation in system development projects in automotive industry. For this purpose an exploratory case study was designed and carried out at Volvo Car Corporation (VCC). Eleven interviews were conducted involving both functional and project managers and other professionals in the Electrical and Electronic Systems Engineering Unit at VCC.

2. METHODOLOGY

To explore the area of cost estimation concerning software in system development projects in automotive industry the overall approach of this study was chosen to be qualitative one.

2.1 Case Study design and research approach

The study was conducted as an exploratory single case study as the research question is of an exploratory nature and the study focuses on contemporary events, without any possibility to manipulate behavior directly, precisely and systematically.

According to the book written by Yin [22], six sources of evidence are most commonly used in case studies, namely documentation, archival records, interviews, direct observations, participant observations and physical events. The *interviews* were chosen as the primary source of evidence. *Direct observation* view was provided by one of the researchers who spent one week field trip at the company and *participant observation* input was added by the other researcher working at the company. However, the *archival records* (time spent on projects, costs etc) were found to be insufficient due to unreliable time reports and the *documentation* did not offer any kind of information that could help triangulate the results. In order to increase the reliability of the results the investigator triangulation was done during the data collection and data analysis phases.

2.2 Data collection

2.2.1 Scheduling

The selection of interviewees was made with help of senior researchers with the knowledge of VCC's organization as well as industry partners working at the company. When the interviewees were selected they were contacted by the researchers as well as their supervisors, with the request to take the time to contribute to the study by participating in the interviews.

2.2.2 Collecting the background information

To better understand the interviewees and their jargon one of the researchers spent a week at the company trying to learn as much as possible about the company culture. The other researcher involved in the study has worked at VCC's unit of Manufacturing Engineering for 8 years.

2.2.3 Interview guide

It was decided at an early stage that the interviews held to collect data would be semi structured in order to allow two-way communication needed for this exploratory study. The interview guide was reviewed by the senior researchers on several occasions in order to assure the quality of the questions content and by industry partners in order to assure that the questions were easy to understand. It was decided that the interview guide would be un-detailed, used more as a reminder of areas to focus on than as a form that must be strictly followed.

The interviewees were asked to speak freely about each focus area and more detailed questions were only asked if the interviewee had not covered them already. In this way it was assured that the interviewees talked about the focus areas as they perceive them and could put more time and effort into focusing on issues that they really believe are important.

2.2.4 Interviews

Due to the explorative nature of the study the data were collected using semi structured interviews. This allowed the interviewers to follow up on interesting leads and understand the problem from the interviewee's perspective.

Interviewees received a presentation of the study in advance, including some questions from the interview guide so that they would feel more confident before and during the interview.

The interviews were conducted by two researchers. One had the role of listener and interviewer, and the other wrote down the interview. Hove and Anda [9] state that interviewees talk more if two interviewers are present and that the amount of follow up questions is greater as well. This leads to more information being gained during the interview.

During the interview the interviewee was encouraged to speak freely and talk about certain situations using follow-up questions as suggested by Hove and Anda [9]. Some time was taken at the beginning and the end of each interview to reassure the interviewee of the anonymity and explain how the data collected during their interview will be handled. The interviewees were told about the objectives of the study and how the results would be presented.

As the issues related to economy are known to be sensitive, according to Hove and Anda [9], the interviews were not recorded so that the interviewees would feel more comfortable. The most sensitive questions were asked late in the interview with the hope that, by then, the interviewee would have gained trust in the interviewers. The focus was not on the exact figures, taking the edge off the most sensitive questions.

Directly after the interviews, the data were reviewed and the interviews were summarized by the two researchers to gain as much useful data as possible.

2.3 Data analysis

To increase the validity of the results the data were analyzed separately by two researchers. Each read the summarized interviews and categorized the data found into facts and "opinions". The facts were removed and used as background information only, and the rest of the data were categorized in

several steps, leading to one list of issues produced by each researcher. The two lists were almost identical. Most of the differences were of a syntax type, and in one case the difference was in the level of abstraction of the issues found. This resulted in two issues being found by one researcher that could be summarized into one issue found by the other researcher. A final list was identified that contained 15 issues.

Results were discussed and reviewed by senior researchers in order to verify the quality of the work done and its findings. A group of industry partners was shown the list in order to verify that the issues were not misunderstood. All the steps were documented to ensure traceability.

2.4 Research Question

This study was designed to explore the following question:

What underlying factors affect software cost estimation in system development projects in automotive industry?

Six focus areas were developed to answer the research question, see Table 1.

Table 1. Focus areas

Focus areas
1. Cost and time estimating procedures
2. Factors that affect the individual estimates
3. People impacting the estimates
4. Usage areas of the estimates
5. Modification of estimates
6. Efficiency of the estimation work

To answer the research question different phases where estimates are made, updated and used needed to be explored as well as personal impacts and opinions of those who make estimates or are affected by them in other ways.

The purpose of *Cost and time estimating procedures* area is to find out how the estimation work is done, if any kind of methodology was used with any templates to be followed, if the people involved in making estimates find the methodology and the templates helpful and how the estimates are reported and to whom.

The *Factors that affect the individual estimates* area focuses on the work done by the individual and factors that affect this work. The focus was on finding out whether those who work with estimates discuss their estimates with anyone, how those people affect the estimators' decisions and whether there are other factors involved than calculations made by interviewees or others.

The *People impacting the estimates* area focuses on the groups dealing with the estimates (if such groups exist). The purpose was to find out how the organizational structure affects the estimates, whether there are groups that make/review the estimates or if it is one-man job. If such groups exist the focus was to be on exploring the group culture, communication and the priority the estimates were given during their group meetings.

The purpose of *Usage areas of the estimates* focus area was to find out if and how the estimates are used, and if they are considered to be useful.

The purpose of *Modification of estimates* area was to understand whether and how the estimates are updated, whether there are formal occasions for doing this and, if so, how the estimators felt about those meetings.

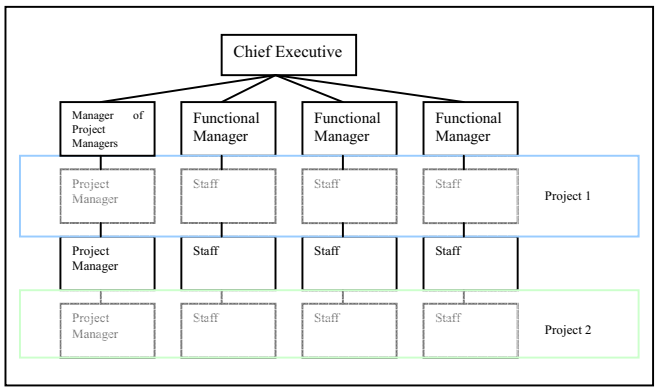
The last focus area, *Efficiency of the estimation work*, focuses on the time that the estimation work takes versus the usefulness of the final estimates, including their correctness.

3. THE CASE OF VOLVO CAR CORPORATION

The study that this paper is based on took place at Volvo Car Corporation (VCC), a car company that was long a part of the Volvo group. In 1999 VCC was sold to Ford Motor Company (FMC). Since then, VCC has been producing 450 000 cars per year, where safety has been made their trademark.

The matrix organization, see table 2, of VCC needed to be considered because the functional and the project parts of the organization have different budgets. The budget for functional part of the organization includes component costs; development costs are included in the project budget. The interviewees were chosen in order to mirror the organization. Nine of them were part of the project organization, ranging from concept phase managers to development managers as well as managers at high and low levels. Two of the interviewees were chosen to represent the functional part of the organization.

Table 2. Matrix organization



The interviews were conducted during the late spring of 2007 and included professionals with long experience in the company and good familiarity with the external factors that could affect the work of estimating time and cost needed to complete the project, such as contact with suppliers and the owner of VCC, FMC.

4. COST ESTIMATION AT VOLVO CAR CORPORATION

Among the data from the interviews some parts were classified as "facts", focusing on the estimation techniques and approaches used by the company. The interviewees stated that they use *expert judgment* [10] approach and to some extent also *analogy* [21] to

past projects. Both *top-down* and *bottom-up* [12] estimates are made in order to get the most realistic cost estimate possible when making estimates for new products.

The interviewees, who all belong to the Electrical and Electronics Systems Engineering Unit, also state that the understanding of how software is developed is inadequate at the other parts of the company, leading to many of the issues mentioned below. Volvo Car Corporation (VCC) is a company that has, as well as the rest of automotive industry, relied mainly on mechanic components for many years. The processes at the company have not yet been fully adapted to the increasing development of software and electronic components.

Final list of issues identified contains 15 issues:

1. *There are problems with comparing estimates and results.* Because time reporting is inadequate there are difficulties in comparing estimates with results.
2. *The estimates are affected by other projects' scheduling.* More urgent projects are given higher priority, which affects the estimates and schedules of later projects.
3. *The opinions about time spent on estimates vary from too much to too little.* Some of the interviewees state that more time needs to be spent on estimates to make them more accurate. Others believe that the development work must be done in any case, and that too much time spent on estimates is a waste of time.
4. *Management estimation goals are not taken seriously. T*he estimation errors that the interviewees state are considered acceptable do not respond to management goals.
5. *Unclear requirements* make it difficult to know what is to be developed and estimated.
6. *Late changes*. These are difficult to foresee, making estimation work more difficult.
7. *The padding is removed from the estimates when detected by management.* This is to be compensated by padding added at higher level of organization hierarchy.
8. *Unrealistic estimates are presented at the gates in order to be able to continue with the project.* As mentioned in issue 7, there is a belief that the development work must be done in any case.
9. *The estimates are affected by the budget and management goals,* such as cost savings and efficiency demands, leading sometimes to too optimistic estimates.
10. *Different company cultures lead to difficulties when communicating estimates to the owner of VCC, FMC (Ford Motor Company).* FMC is the owner of VCC, and cooperation between the companies is aggravated by different cultures in terms of development, management and politics.
11. *There is a lack of competence in assessing suppliers' estimates and resources.* As mentioned in issue 3, there is some in-house development and the interviewees state that this should be used when reviewing prices proposed by suppliers, as well as when estimating suppliers' abilities to finalize the project successfully. The company has a team that revises cost estimates for hardware components developed by suppliers. However, no such service is provided for software development, making it difficult to understand whether prices there are reasonable or not.
12. *Opinions on what template there is for the purpose of cost estimation differ,* from no template at all to different kinds of documents that could be used to provide support while making estimates.
13. *There is a lack of coordination between functional and project parts of the organization during the early development phases with regard to estimates.* The interviewees state that, because there are separate budgets in the two parts of the organization, there is a lack of cooperation between them, making estimation work more difficult.
14. *The existing estimation competence is not used by the company.* The interviewees state that there is in-house development of software components and that it would be preferable to use this competence when making estimates.
15. *There are problems estimating new technology.* This is a well known problem; it is always easier to make estimates for familiar components.

4.1 Analysis of the results

To put the results of this study in a broader perspective, they will be compared to the results of the study done by Lederer and Prasad [15] [16] exploring the issues that affect the process of cost estimation. The study was done in the form of a questionnaire, answered by 112 software professionals, resulting in 16 factors correlated to estimation inaccuracy. There are other studies done in this area that could have been used for this purpose, such as the study by Phan et a. [18], the van Genuchten study [6], the Subramanian and Breslawski study [20] or the Standish group report [19]. However, taking in consideration factors such as the date of publication, presentation and detail level of the result and methods used the Lederer and Prasad study [15] [16] was found to be the most suitable for this purpose.

The issues found in the study underlying this paper were categorized in the same manner as in the Lederer and Prasad paper focusing on causes of inaccurate cost estimates [15], in the four categories:

- Management issues dealing with project control.
- User communication issues
- Politics dealing with the issues such as manipulating real costs in order to meet the estimate.
- Methodology issues considering the issues that affect the tuning of the estimate.

Table 3. Management issues compared to issues found by Lederer and Prasad [15]

Management issues found	Comparison to literature
1. There are problems with comparing estimates and results.	I18. Performance reviews don't consider whether estimates were met I8. Lack of project control comparing estimates and actual performance (I12. Inability to tell where past estimates failed)
2. The estimates are affected by other projects scheduling.	-
3. The opinions on time spent on estimates vary from *too much* to *too little*.	-
4. Management estimation goals are not taken seriously.	-
-	I 24. Lack of careful examination of the estimate by Information Systems Department management

Among the issues in the *Management* category presented in table 3, one is confirmed by the Lederer and Prasad study [15], namely issue 1, *There are problems with comparing estimates and results*. The interviewees mentioned the problems brought up by Lederer and Prasad [15] in their issue 8, *Lack of project control comparing estimates and actual performance*, issue 18, *Performance reviews don't consider whether estimates were met*, and to some extent issue 12, *Inability to tell where past estimates failed*.
Issues 2, *The estimates are affected by other projects scheduling*, 3, *The opinions on time spent on estimates vary from too much to too little*, and 4, *Management estimation goals are not taken seriously*, found in the study underlying this paper are however not found in the literature.

Table 4. User Communication issues compared to issues found bu Lederer and Prasad [15]

User communication issues found	Comparison to the literature
5. Unclear requirements.	I10. Poor or imprecise problem definition
-	I17. Users' lack of understanding of their own requirements
-	I14. Frequent request for changes by users
-	I16. Users' lack of data processing understanding

Clear differences are found among the *User communication issues* presented in table 4, mostly depending on the fact that there is no clear user in the car industry that is present during the development, as there might be in the software industry. The issue 5 found in the study this paper is based on, *Unclear requirements,* is confirmed by the issue 10, of the Lederer and Prasad study [15].

Table 5. Politic issues compared to issues found by Lederer and Prasad [15]

Politic issues found	Comparison to literature
6. Late changes.	I21. Reduction of project scope or quality to stay within estimate, resulting in extra work later (I14. Frequent request for changes by users)
7. The padding is removed from the estimates when detected by management.	I20. Removal of padding from estimate by manager
8. Optimistic estimates are presented at the gates, in order to be able to continue with the project.	I22. Pressures from managers, users or others to increase or reduce the estimate
9. The estimates are affected by the budget and management goals.	I22. Pressures from managers, users or others to increase or reduce the estimate
10. Different company cultures lead to difficulties when communicating estimates to the owner of VCC, FMC.	-
11. There is a lack of competence in assessing suppliers' estimates and resources.	-
-	I19. Red tape (paperwork)

Even though the issues found in this study are formulated in slightly different way in order to truthfully reflect the data collected during the interviews issue 6, *Late changes,* issue 7, *The padding is removed from estimates when detected by management*, issue 8, *Optimistic estimates are presented at the gates, in order to be able to continue with the project*, and issue 9, *The estimates are affected by the budget and management goals*, are confirmed by the issues found by Lederer and Prasad [15]. Two new issues are however listed among the *Politic issues* presented in table 5, namely issue 10, *Different company cultures lead to difficulties when cooperating with the owner (FMC)* and issue 11, *There is lack of competence in assessing suppliers' estimates and resources*.

Table 6. Methodology issues compared to issues found by Lederer and Prasad [15]

Methodology issues found	Comparison to literature
12. The opinions on what template there is for the purpose of cost estimation differ.	I 3. Lack of adequate methodology or guidelines for estimating
13. There is a lack of coordination between the functional and project parts of the organization during the early development phases with respect to estimates.	I 4. Lack of coordination of systems development, technical services, operations, data administration etc. functions during the development.
14. The existing estimation competence is not used by the company.	I 12. Inability to tell where past estimates failed I 13. Insufficient analysis when developing the estimate
15. There are problems estimating new technology.	-
-	I 5. Lack of setting and review of standard durations for use in estimating

Issue 12 of this study, *The opinions on what template there is for the purpose of cost estimation differ*, is compared to Lederer and Prasad's issue 3 [15]. However, the interviewees did not mention that they lack methodology, and they seamed to be fairly certain how to make estimates. The problem seems to be that they use different, more or less official templates, if any. There seems to be no common template, or if there is such a template, there have been difficulties with communicating it to the employees. All the issues found in this category were confirmed by the literature. While issue 4 is not mentioned as an issue in the Lederer and Prasad study [15], however, it is a well known risk often mentioned in the literature [2].

4.2 New issues

The findings of this study imply that the list of issues found by Lederer and Prasad [15] needs to be extended with 6 issues to better mirror the case of automotive industry:

I 15: There are problems estimating new technology.

I 2: The estimates are affected by other projects scheduling.

I 3: The opinions on time spent on estimates vary from *too much to too little*.

I 4: Management estimation goals are not taken seriously.

I 10: Different company cultures lead to difficulties when cooperating with FMC.

I 11: There is a lack of competence in assessing suppliers' estimates and resources.

Issue 15, *There are problems estimating new technology,* is a well known risk that can found in the literature, it is for example mentioned in a Boehm paper [2].

Issue 2, *Estimates are affected by other projects scheduling*, issue 3, *The opinions on time spent on estimates vary from* too much *to* too little and issue 4, *Management estimation goals are not taken seriously* could be compared with problems described in van Genuchten paper [6]: *No commitment by personnel to the plan* and *Priority shifts*.

The problematic relationship between vendor and supplier, mentioned in issue 11, has been discussed by many researchers. Bruce et al [4] describe in their article both risks and negative experiences associated with such collaborations. They mention more costly and complicated development, loss control ownership and differing aims and objectives leading to conflicts.

The main product of VCC is not software, even though the amount of software in cars is growing. Issue 10, unique to this study, and issue 11 focus on two interesting issues that need to be addressed; the effect of the relation between a company and its owner and the relation between a company and its suppliers.

5. DISCUSSION OF RESULTS AND VALIDITY

The objective of the study that this paper is based on was to elicit the factors that affect software cost estimation in system development projects in the automotive industry. The paper focuses on the factors that affect the prerequisites for successful software cost estimation in systems development projects within the automotive industry.

The results presented in this paper were compared to a study exploring the issues that affect the cost estimation in software development projects done by Lederer and Prasad [15]. Many of the issues in this paper are confirmed by some of the highest ranked issues found in the Lederer and Prasad study [15]. However, the study underlying this paper was carried out at a car company which affects the results. The profit margins are smaller and the projects are more time critical than they might be at a typical software company. Also, VCC being a company with a long history of primarily mechanical development could also be a factor that affects the results.

Looking at the *Methodology issues* it can be seen that the prerequisites for successfully using the methodologies mentioned by the interviewees are insufficient. To make successful *expert judgment* [10], the experts should be used. This is contradicted by the issue 14, *The existing estimation competence is not used by the company*. In order to perform analogy to earlier projects, those projects must be documented, both as concerns the results and estimates. However, issue 1 states that *there are problems with comparing estimates and results*. Not being able to compare estimates with the results makes it impossible to understand which (if any) parts of the estimates were too optimistic/pessimistic (see issue 1). Inadequate time reports by the developers, not reporting which projects they spend time on, make it impossible to know how much each project actually costs.

To make estimates as accurate as possible including prices of suppliers, the functional and project parts of the organization must be able to cooperate. However, cooperation is complicated by their budgets being separated which might be leading to competition instead of cooperation.

Delays in other projects might lead to changes in plans and estimates (see issue 2), more or less permanent, making the comparison of early estimates and results even harder. The fact that the developers seem to primarily be concerned with developing safe, high quality components could lead to issue 4

which states that *management estimation goals are not taken seriously*.

The interviewees state that the development work must be done properly, no matter what the situation. New cars must be developed and produced. This attitude, together with management goals of cost savings and greater efficiency could lead to producing optimistic estimates in order to be able to continue with the project. The management goals of increased effectiveness and savings are also believed to lead to unrealistic estimates.

The difference in the results in the *User Communication* category can be explained by the fact that there is no clear end user involvement that can affect the estimates. The end user is considered by the company, and this fact has been taken in to consideration in issue 6. Issue 5, *Unclear requirements*, is an important issue in terms of estimates. Not knowing at an early stage what is to be developed is tantamount to not knowing what is to be estimated and paid for. The issue of unclear requirements is well known; it has been addressed by many researchers in the area of Requirements Engineering.

We believe that the prerequisites for successful cost estimation are an issue that must be explored further in order to decrease the errors in estimates while using any of the wide range of estimation methodologies available.

5.1 Validity discussion

According to Yin [22], to test the validity of results of a study the construct validity, internal validity, external validity and reliability must be considered.

5.1.1 Construct validity

According to Yin [22] construct validity establishes correct operational measures for the concept being studied. Tree tactics can be used to increase level of construct validity, namely use of multiple sources of evidence, established chain of evidence and having key informants reviewing draft of case study report.

Multiple sources of evidence were used, using investigator triangulation (two evaluators). Two researchers were present during the data collection. During data analysis each of the researchers performed the analysis on their own and the results were compared afterwards. One of the researchers has been working at the VCC's unit of Manufacturing Engineering for 8 years. However, the unit of Manufacturing Engineering has not been a part of the study; the study was performed at the Electronics development department.

The *chain of evidence* was established and maintained from the case study questions and protocol to summarized interviews, citations that were saved in the database connecting them in various steps to the study report and this paper.

The *key informants*, in this case both peers and representatives from the industry followed the study and were invited to review the results before publication. The reports were written in a way that did not disclose the identity of the participants in the study.

5.1.2 Internal validity

According to Yin [22], the internal validity an important issue mainly in explanatory studies. Internal validity is increased by establishing a causal relationship where certain conditions lead to other conditions. In the study presented here the problem of interviewee not feeling comfortable in talking about the sensitive issues such as presenting unrealistic estimates could be discussed. To increase interviewees' trust the interviews were not recorded. At the beginning of each interview the interviewer took the time to ensure the interviewee of his/her anonymity and explain exactly how the data from the study would be handled (kept in a safe and encrypted form while saved in the computers). All the questions the interviewees might have had were carefully answered.

5.1.3 External validity

To increase external validity the domain to which a study's findings can be generalized should be established, according to Yin [22]. The tactics that could be used for this are using theory in single-case studied and replication logic in multiple-case studies.

This study is a part of a larger multiple case study. The results will be easier to generalize for the vehicle industry projects upon completion of the final study.

However, in order to assure the external validity of this particular case the results of the similar study done by Lederer and Prasad [15] were used to confirm the results. Many similarities are found leading to believe that the generalization might be possible.

5.1.4 Reliability

To increase reliability it must be ensured that the study can be repeated, with the same results, according to Yin [22].

The case study was carefully documented in order to make it possible for it to be replicated later on. All the data have been stored, linking the case study protocol with interview summaries, citations and the *results database*.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The purpose of the study that this paper is based on was to explore whether there are any underlying factors that affect software cost estimation in systems development projects in the automotive industry. Fifteen such factors were found in this study, categorized as *Methodology issues*, *Management issues*, *Politic issues* and *User communication* issues.

The findings were compared to the existing literature, a study done by Lederer and Prasad [15], presenting the findings of a study of 112 information system managers and other professionals. There are issues presented in this paper that are not found in the Lederer and Prasad study [15], such as ones saying that estimates are affected by the budget, projects schedules and management goals not being taken seriously. There are also issues concerning company cultures and difficulties when cooperating with the owner of VCC (FMC) and suppliers in regard to estimates.

The differences found could be caused by the fact that this study was carried out at an automotive company in which the amount of software components is growing in number and complexity, but is still in minority as compared to other kind of components. Therefore, the issues found in the study underlying this paper are only to be considered in the automotive domain.

6.2 Future work

Being a part of a larger, global company affects the process of estimation. We believe that this is an interesting issue to explore,

as well as the effects of the relationship with suppliers due to the fact that much of the development work concerning electronic and software components is done by the suppliers.

7. ACKNOWLEDGMENTS

This research has been conducted within the COSY project which is funded by the Swedish industry and government joint research program IVSS – Intelligent Vehicle Safety Systems.
We would like to thank the Volvo Car Corporation for giving us the opportunity to conduct this study, and the interviewees that participated in the study for taking time from their busy schedules in order to help us.
Also, we would like to thank associated professor Sofia Börjesson at Technology, Management and Economics department at Chalmers University of Technology for valuable input during the phases of preparation, analysis and reporting and PhD students at the department of Software engineering for the feedback during the writing of this paper.

8. REFERENCES

[1] Boehm, B.W. 1991. Software risk management: principles and practices. In Software, IEEE. **8**(1): p.32-41

[2] Boehm, B.W., Clark, B., et al. 1996. The COCOMO 2.0 software cost estimation model: A status report. In American Programmer. **9**(7): p.2-17

[3] Broy, M. 2006. Challenges in Automotive Software Engineering. In Proceeding of the 28th International Conference on Software Engineering, ICSE '06. p.33-42

[4] Bruce, M., Leverick F., Littler D., Wilson D. 1995. Success factors for collaborative product development: a study of suppliers of information and communication technology. R&D Management. **25** (1): p. 33–44.

[5] Grimm, K. 2003. Software Technology in an Automotive Company – Major Challenges. In Proceedings of the 25th International Conference on Software Engineering, IEEE 2003. p. 498

[6] van Genuchten, M. 1991. Why is Software Late? An Empirical Study of Reasons for Delay in Software Development. In Transactions on Software Engineering. **17**(6), p. 582-590

[7] Heemstra, F.J. 1992. Software cost estimation in Information and Software Technology. **34**(10): p. 627-639.

[8] Hill, A., Thomas, L.C., Allen, D.E 2000. Expert's estimates of task durations in software development projects. In International Journal of Project Management. **18**(1), p. 13-21

[9] Hove, S.E., Anda, B. 2005. Experiences from conducting semi-structured interviews in empirical software engineering. In 11th IEEE International Software Metrics Symposium (METRICS´05). p. 23

[10] Hughes, R.T. 1996. Expert judgment as an estimating method. In Information and Software Technology. **38**(2), p. 67-75

[11] Jongmoon, B., Boehm, B., et al. 2002. Disaggregating and calibrating the CASE tool variable in COCOMO II. In IEEE Transactions on Software Engineering. **28**(11): p. 1009-1022

[12] Jorgensen, M. 2004. Top-down and bottom-up expert estimation of software development effort. In Information and Software Technology. **46**(1): p.3-16

[13] Jorgensen, M., Sheppard, M. 2007. A Systematic Review of Software Development Cost Estimation Studies. In IEEE Transactions on Software Engineering. **33**(1): p. 33-53

[14] Kitchenham, B. 1996. Software Metrics: Measurement for Software Process Improvement. Blackwell Publishers.

[15] Lederer, A.L., Prasad, J. 1995. Causes of Inaccurate Software Development Cost Estimates. In Journal of Systems and Software. **31**(2): p. 125-134

[16] Lederer, A.L. and Prasad, J. 1993. Information systems software cost estimating: a current assessment. In Journal of Information Technology. **8**(1): p. 22-33.

[17] Moores, T.T., Edwards J.S. 1992. Could large UK corporations and computing companies use software cost estimating tools? – a survey. In European Journal of Information Systems. **1**(5): p. 311-319

[18] Phan, D., Vogel, D., Nunamaker, J. 1988. The Search for Perfect Project Management. In Computerworld. p. 95-100

[19] Standish group. 1994. Can be fount at http://www.standishgroup.com/sample_research/chaos_1994_1.php (2007-10-12, 16:14)

[20] Subramanian, G.H., Breslawski, S. 1995. An empirical analysis of software effort estimate alterations. In Journal of Systems and Software. **31**(2): p. 135-141

[21] Walkerden, F., Jeffery, R. 1999. An Empirical Study of Analogy-based Software Effort Estimation. In Empirical Software Engineering. **4**, p. 135-158

[22] Yin R. 2003. Case study research: design and methods. 3rd edition. SAGE Publications, Inc. Thousand Oaks

A case study of the interaction between development and manufacturing organizations with a focus on software engineering in automotive industry

Joakim Pernstål
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
+46(0)31 325 02 42
pernstal@chalmers.se

Ana Magazinovic
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
+46(0)31 772 57 35
ana.magazinovic@chalmers.se

Peter Öhman
Chalmers
Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden
+46(0)31 772 36 64
peter.ohman@chalmers.se

ABSTRACT

As most future automotive innovations will be realized with software, the automotive industry is facing a transition from mechanical to software engineering. To achieve successful product launches, this is also true for manufacturing engineering.

The purpose of this study is to gain a better understanding of the interaction between the organizations for Research&Development and manufacturing, specifically focusing on the development of software and electronics in the automotive industry. It is presented as a case study with a qualitative approach where data were collected from documents and in interviews with practioners at a Swedish automotive company. Three main strategies were used to obtain validity of the results: 1) prolonged involvement, 2) triangulation and 3) peer debriefing.

It can be concluded that there are challenges in the research area since 24 issues emerged from the data. The results may primarily be used as input for improvements within the company and the methodology can be utilized by other organizations that are interested in founding their development of work practice on empirically observed findings.

Categories and Subject Descriptors

K.2.6 [**Software Engineering**]: Software Management – S*oftware development, Software process*

General Terms

Management, Documentation, Design, Verification

Keywords

Empirical Software Engineering, Case Study, Automotive, Manufacturing.

1. INTRODUCTION

Trends in the automotive industry show that the number of electrical functions in vehicles is increasing and that they are becoming more complex Grimm [1] and Broy [2]. Current vehicles contain a number of Electronic Control Units (ECU) with the prerequisite software that controls various electrical systems such as Infotainment and Climate Control. These systems are mainly realized by software and electronics and Grimm [1] estimates that 80 percent of all future automotive innovations will be driven by electronics and 90 percent thereof by software. According to Broy et al. [9] current premium cars contain about 100 MB of binary code and it is expected that the upper class vehicles will have up to 1 GB of software in five years time.

This evolution is pressing the automotive industry to elaborate appropriate processes for the development of software based systems. The main goal of these processes is to achieve a quality assured and cost efficient launch of the vehicles in the manufacturing process when production starts. Further, customers' demands for a broader range of products with various functions and tighter scheduling of new model launches involve an increased number of products and systems that must be developed in a shorter time.

Consequently, the involvement of manufacturing engineering in development processes has become imperative since the increase in software based systems implies a greater number of product variants and a higher degree of complexity that has to be managed in the manufacturing processes. Product related items that influence production are for instance electrical architecture, software download and diagnostic concepts and file sizes. Moreover, to ensure that all the systems in the car work correctly when the car leaves the production line, methods for tests and verification of functions and components that fit production have to be designed during the development processes. Thus, to achieve a successful launch, the product must be harmonized with the prerequisites for the manufacturing processes, e.g. line speed, tools, competence and man power which requires a well functioning interaction between the organizations for manufacturing and Research&Development (R&D).

This paper presents an exploratory study whose purpose was to gain a better understanding of the interaction between the organizations for R&D and Manufacturing, specifically focusing on the development of software based systems in the automotive industry, from the point of view of stakeholders in the studied organization. The authors expect that there are challenges related to the interaction between manufacturing and R&D during

development of software based systems in vehicles. The increase of software in vehicles and the experience and knowledge of one of the researchers provide the grounds for proposing this theory.

The research was done as an explorative case study at Volvo Car Corporation (VCC) where one of the researchers has worked with electrical manufacturing engineering for eight years. Twelve interviews were conducted with different stakeholders concerned with software development, who were selected in order to constitute a representative sample for the research area. The outcome of this study will be a number of findings that sets the outlines for future research.

The paper is organized as follows. Section 2 covers earlier work related to the topic of this study. Sections 3 and 4 describe the method and the analysis of data. Section 5 presents the results and sections 6 and 7 discuss the credibility of this work and the results. Finally, conclusions and future work are presented in section 8.

2. Related work

Research covering the specific research area dealt with in this work seems limited and especially when it comes to empirically grounded work. However, Grimm [1], Broy [2] and Broy et al. [9] support the assumed trend of a rapid increase in software and software based functionality in automotives and discuss challenges for the automotive industry. Particularly Broy [2] presents challenges such as the need of building up software competencies and improving and adapting development processes so that they support software engineering. Moreover, Broy [2] discusses the importance of requirement engineering, systems integration and maintenance e.g. compatibility, diagnosis and repair, which are issues that have an impact on the interaction between R&D and manufacturing.

There are a number of software engineering studies addressing other research areas, which are using a qualitative approach where the views from different stakeholders are considered. One example is the study by Jönsson and Wohlin [10] which investigates how potential issues and uses associated with impact analysis are seen on three organizational levels. Another example is the paper by Berling and Höst [8] where the characteristics of verification and validation activities in the software development process are studied by analyzing the views from testers and test managers.

3. Methodology

The overall research method applied here is a case study as described in Yin [3]. Case studies can be exploratory, descriptive or explanatory. The lack of previous empirical case studies in the area of interest motivates the explorative nature of this study. Case studies are most suitable for investigating research questions of the types what, how and why. The purpose of this study is to gain a better understanding of the interaction between the organizations for R&D and manufacturing, specifically focusing on development of software based systems in the automotive industry. Hence, the research question is

What challenges are there in the interaction between R&D and Manufacturing, specifically focusing on the development of software based systems in the automotive industry?

The study was conducted using a qualitative research approach since it, according to Robson [4], is useful when the purpose is to explore an area of interest, to obtain an overview of a complex area, and to discover diversities and variety rather than similarities. This approach was found appropriate because of the explorative nature of the study with the objective to use the results for eliciting theories/hypotheses and set the baseline for future research.

3.1 Case description

This work was conducted at VCC and studied incorporated processes in the Electrical Development Process (EDP). EDP originates from the traditional V-model and follows the overall product development system, with its milestones (gates) for decision making in a vehicle project.

One key process identified that comprises frequent interactions between R&D and manufacturing is the software release process which manages the release of software from product development to manufacturing.

The strategy for sampling interviewees can be described as heterogeneous and purposive, see Robson [4]. Thus, the objective of selecting participants was to pinpoint and cover all the roles that are involved in the interaction between manufacturing and R&D. This was achieved by examining the roles in the software release process together with representatives of the studied organization. Table 1 gives a concise description of the selected key roles.

Table 1. Description of selected key roles

Role	Org	Description
Designer	R&D	This role is responsible for developing software and hardware that fulfils technique, time, and cost
Project Leader (PL)	R&D	This role is responsible for the electrical content in a car project.
Configuration Manager (CM)	R&D	This role is responsible for that new software files fulfils requirements for release in production.
Manufacturing Engineer (ME)	Man*	This role is responsible for development and implementations of manufacturing processes for software download and test/verification of vehicles

* Manufacturing

Additionally, documents describing software development processes and archival record with project information at the company were used.

3.2 Case study design

In Yin [3] four types of case study designs are discussed. This study can be characterized as a single case study with four embedded units of analysis which consist of the key roles sampled. The rationales for the chosen design are based on the fact that one of the researchers is familiar with the case under study and has access to the organization. Further, the aim of this study is to explore the area of interest and the results will be used as a baseline for future research on similar cases.

3.3 Data collection

Six data sources that are most commonly used in case studies are discussed in Yin [3]: documentation, archival records, interviews, direct observation, participant-observation and physical artifacts. The data sources used and found most appropriate for this study were interviews with identified key roles and pertinent documentation and archival records at the company.

3.3.1 Planning and preparations

The participants were selected on the basis of an expert judgment by one of the researchers together with company representatives. The main objective of the selection of participants was that they should represent all the key roles identified. Further, it was preferred that the number of participants in each role would be fairly balanced. However, a balanced distribution was difficult to obtain since the availability of staff for some roles was limited. This imbalance was alleviated by selecting three participants at R&D with experience of manufacturing engineering. Table 2 shows the distribution of participants.

Table 2. Participant distribution

R&D			Manufacturing
Designer	**PL**	**CM**	**ME**
5	4	1	2

The organization and the participants were informed about the study and the measures taken, such as data protection, for integrity of the organization and participants. The interviewees were guaranteed full anonymity.

The risks for biases from one of the researchers who works in the studied environment and is a close colleague to some of the participants were treated as a major concern during the preparation. It was difficult to find literature that gives guidance for this type of research situation. In Robson [4] it is recommended to work in teams whenever possible because of the advantages in sharing and assessing data. Further, Berling and Höst [8] deals with a similar situation by writing a tentative model of the work place in the preparation phase. This was not feasible here, however, owing to the magnitude and complexity of the case studied. To tackle the risks for biases on the part of the researcher, it was agreed to assist the research project with necessary resources from the research community. Moreover, it was also decided that pre-interviews would be conducted where the bias from the researcher with a close relation to some of the participants was evaluated.

Another issue that was discussed during the planning was the number of interviewers. Little literature discussing this issue for qualitative research within software engineering was found. However, Hove and Anda [5] presents a number of advantages and disadvantages of having two interviewers. Based on this and practical circumstances it was decided to conduct the interviews with two interviewers, where one was responsible for the interview process and the other took extensive notes. Moreover, the use of two interviewers involves two observers which would enhance the credibility of the results and reduce the risk discussed above, regarding biases on the part of one of the researchers.

It was also discussed whether the use of tape a recorder would prevent the participants from expressing their real opinions. Advantages and disadvantages of recording are discussed in Hove and Anda [5] who strongly recommend this technique for research in software development. However, as the risk for the participants' unwillingness to talk freely and openly was considered a major threat to this enquiry, it was decided to mitigate this risk by conducting pre-interviews and evaluating the usage of a tape recorder.

The interview questions were developed by the researchers and designed to cover the area of interest and answer the research question. The questions were designed as open-ended, supporting a semi-structured interview style described in Robson [4]. This interview technique was found most appropriate for this study since it is primarily explorative. Further, in order not to drift away from relevant subjects during the interviews, it is recommended that there be a certain level of structure. An interview guide was prepared and based on the recommendations in Robson [4]

Introduction Interviewer introduces him/herself, explains the purpose, assures confidentiality and asks for relevant background information such as roles, experience and projects.

Warm-up Easy, unthreatening questions are posed at the beginning to settle the interviewee down.

Main body of interview This covers the main purpose of the interview in what the interviewer considers the logical progression. In semi-structured interviewing, this order can be varied, capitalizing on the responses made.

Cool off Usually A few straightforward questions are posed at the end to defuse any tension that might have built up.

Closure Interviewees may, when the recorder is switched off or the notebook is put away, come up with a great deal of interesting material. It is therefore recommended to ask the interviewee whether there is something that has not been covered before closing the interview session.

To ensure that the questions were comprehensible, unambiguous and had the ability to answer the research question, the interview guide was reviewed by members of the research community and representatives of the studied company.

3.3.2 Interviews

A pretest was carried out by interviewing participants representing different roles. The pretest had three major objectives: 1) check the risk for loss of information when using a tape recorder, 2) check bias from one of the researchers when interviewing colleagues and 3) evaluate the interview guide. Four interviews were conducted by two researchers where two of the sessions were recorded. Moreover, one of the interviews was with a close colleague of one of the researchers

The impression was that the interviewees spoke freely and were not bothered by the presence of the tape recorder during the session. Further, Hove and Anda [5] strongly recommends the use of a recording device when software development topics are discussed; thus it was decided that the remaining interviews would be recorded.

The risk for bias from one of the researchers when interviewing a close colleague was judged by the researchers as impossible to exclude. Hence, the researcher was replaced by another researcher, familiar with the study, during further interviews with these colleagues.

The interview guide was slightly modified where the sequence and the formulation of some of the questions were changed. Further, the pre-interviews were considered possible to use in the analysis.

Twelve participants were interviewed on site over a period of one month and all the interviews were held in Swedish. Table 3 presents the length of the interviews in minutes.

Table 3. Interview length in minutes

Interviews	Mean interview time	Std.dev interview time	Total Interview time
12	93	17	1120

4. Data analysis

The analysis was influenced by the principles of grounded theory presented in Glaser and Strauss [6] , i.e. theories grounded in data, since this inquiry has an explorative approach. However, complementary advice was needed as it is not an easy task to carry out an analysis based only on the prescriptions for a genuine grounded theory. In Yin [3] the importance of having a general analytic strategy is advocated, which is the best preparation for conducting a case study as it facilitates the analysis. Analysis of case studies can be based on three strategies: 1) relying on theoretical propositions, 2) rival explanations and 3) case descriptions. The first strategy was chosen for the analysis by relying on the theoretical proposition for this study, which claims that there might exist challenges regarding the interaction between manufacturing and R&D during development of software based systems in vehicles. The data analysis was based on the approach described by Miles and Hubermann [7] with three concurrent "flows of activity": data reduction, data display and conclusion drawing/verification.

4.1 Interview and document analysis

All the interviews were documented with notes taken by one of the interviewers and ten were also recorded. The recorded interviews were transcribed before the analysis and varied between six and ten pages in length. The transcription was done by one of the researchers since it was not possible to allocate extra resources for this time-consuming work. Tapes, transcripts and notes were stored in a case study database as described in Yin [3]. The analysis of interview data was divided into four main phases where all the phases were iterated several times, see figure 4. The first phase included filtering of data by extracting statements from passages in the raw data that could in some way be a potential issue in the area of interest. In the second phase, the remaining information was scrutinized, gathered in a list and sorted into statements with an identification number and a description. Moreover, to be able to trace the statements, the list also contained references to passages in the raw data. Phase 3 comprised grouping statements that were displayed as issues at an appropriate level of abstraction so that they could be linked to theory and research question. To enhance the credibility of the results, these phases were carried out separately and in parallel by two researchers. In phase 4 the two researchers reviewed and discussed their results together with two other researchers, which resulted in a number of refined issues. Further, the issues were sorted into three main categories on level 1 that were based on the sub processes to the product&manufacturing development process at the company. Categories on lower levels emerged from the analysis of data. Figure 1 shows an overview of the interview data analysis.

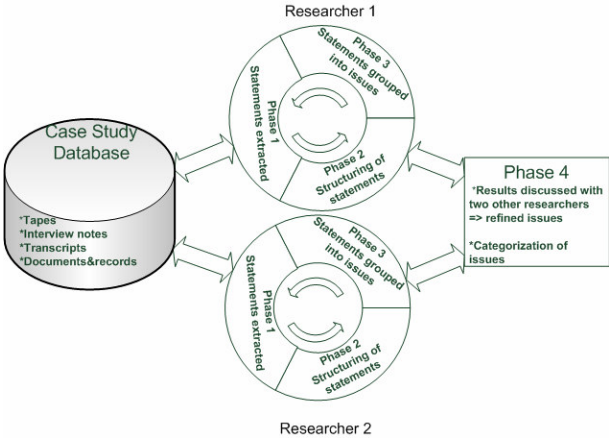


Figure 1. Interview data analysis

The document analysis was guided by the extracted results of the interviews and covered archival records with project information and documentation describing processes and instructions related to EDP. For each of the issues identified, relevant documentation was examined to check whether it indeed confirmed the issue. If the documentation contained what was included in an issue, it was considered to support it. Similarly, if the documentation contradicted what was included, it was considered not to support the issue.

4.1.1 Traceability

According to Yin [3] the benefits of six sources of evidence can be maximized if the study follows three principals: 1) use multiple sources of evidence, 2) create a case study database and 3) maintain a chain of evidence. Both interviews and documents were used as data sources in this study. Collected data and extracted statements made in the interviews about the area of interest were organized and filed in the case study database. Further, remaining data were merged into issues with cross references to tagged passages in interviews and pertinent documents. Since this study has a flexible design, as discussed in Robson [4], research procedures and questions that emerged, were described in a case study protocol based on the recommendations in Yin [3] and attached to each issue together with the initial purpose and research questions in the study. This enhances the possibility to trace any changes concerning procedures and research questions during the inquiry. Table 4 shows the principal layout of the case study database.

Table 4 Principal layout of the case study data base

Issue	Description	Interview statements	Documents /records	Procedures and questions
1	Production requirements are unclear and not easy to find.	Interview 1 statement 2 (1:2); 1:8;1:9…..	Document. xxxx Record yyyy	Interviews and documents, interaction R&D and Manufacturing

5. Results

The analysis resulted in 24 issues divided into two groups. Table 5 contains issues in group 1 that primary address the research question in this study. Secondary issues that were found important but have a weaker relation to the scope of this work were placed in group 2, see table 6. Further, tables 5 and 6

present the contribution from the used information sources and how respective issue was supported by participants at R&D and manufacturing.

Table 5. Primary issues

Issue no	Description	Interview study		Document study
		R&D	Man*	
1	Production requirements are unclear and not easy to find	X	X	X
2	Lack of processes that secure design for testability in production	X	X	X
3	Lack of processes that secure design for SW download and Car Configuration in production	X		X
4	Compliance with processes and instructions for quality assured release of software changes in production can be improved	X	X	
5	Prerequisites for appropriate verification of manufacturing processes that are affected by SW in vehicles are not fulfilled	X	X	X
6	Lack of modeling possibilities of manufacturing processes that are affected by SW in vehicles	X	X	X
7	There are potential improvements of the manufacturing processes that are affected by SW in vehicles	X		X
8	SW-competence can be improved within the manufacturing organization	X	X	
9	Knowledge about manufacturing within the R&D organization is important	X	X	
10	There is a need of preserving and facilitating a smooth collaboration between manufacturing and R&D	X	X	

*Manufacturing

Table 6. Secondary issues

Issue no	Description	Interview study		Document study
		R&D	Man*	
11	Working processes and quality instructions for development of SW and electronics are not always adopted	X	X	
12	Method of working developed and tools tends not to be adapted for development of SW and electronics	X		X
13	Difficult to achieve smooth transfer of new technology to product development projects	X		
14	Deployment of resources is not given priority in early product development phases	X	X	
15	Decisions and decision making processes are not always perceived as clear	X		
16	The quality of specified product requirements needs to be improved	X		X
17	Late changes of product requirements cause difficulties in quality assuring SW functionality	X		
18	Methods for inspection/verification of requirements can be improved	X		X
19	Lack of efficient tools and working processes for requirement handling	X		X
20	Lack of sufficient tools and working processes for modeling SW functionality.	X		
21	Appropriate conditions for supporting and communicating with supplier are essential for the result	X		X
22	Collaboration with other brands and internal between functions/disciplines can be improved	X	X	
23	Lack of understanding for SW engineering outside the units working with SW	X	X	
24	The suppliers' experience and competence are vital for the quality of deliveries	X		

*Manufacturing

Figure 2 shows the categorization of the issues where level 1 has been derived from the sub processes to the product&manufacturing development process. Underlying levels emerged from the data set.

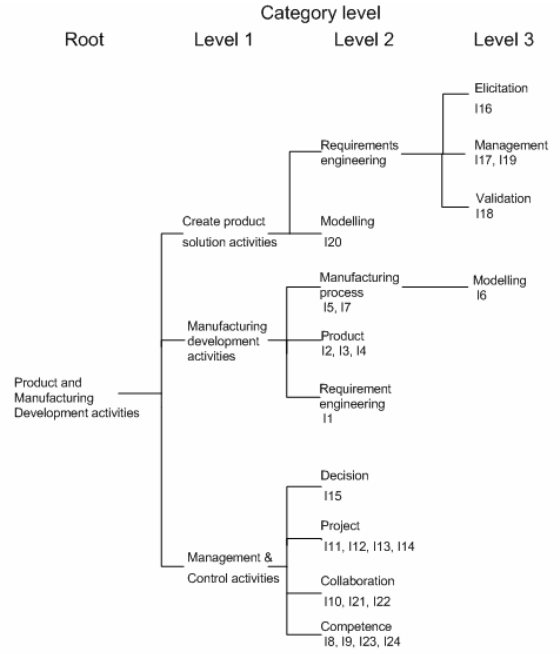


Figure 2. Categorization of issues

The following subsections present and discuss the primary issues in table 5 and provide brief comments on the secondary issues in table 6. Discussions concerning the validity and the results are provided in sections 6 and 7.

5.1 Primary issues

Issue 1 Production requirements are unclear and not easy to find:

This issue was agreed upon by all the participants. The general opinion was that there is a lack of explicit production requirements particularly considering diagnostics and available time for software download. It was also mentioned that an official way of working in handling prerequisites for production is missing and is mainly based on experience and informal contacts. One comment was "Production requirements is just something you hear about or learn the hard way" Hence, experience and a sufficient organizational network seem essential to managing a flawless launch of developed software in production.

Issue 2 Lack of processes that secure design for testability in production:

Design for testability of software based functions in production means that test methods for functions and components are developed to meet the manufacturing demands on efficiency and delivered quality. For example, a visual test by an operator does not fulfill these goals since the test method is subjective (judgment of an operator) and requires operator time.

Missing working procedures during product development for dealing with elicitation of test methods in production was a recurrent comment among the interviewees. They particularly mentioned the need for developing and tailoring diagnostics for manufacturing purposes as well. However, the participants had deviating views about the organizational aspects, specifically regarding how responsibility should be distributed between manufacturing and R&D for driving this task. Table 7 shows the distribution between the interviewees who commented on this matter and their organizational affiliation.

Table 7. Distribution of statements on responsibility for developing test methods in production

Participant affiliation	R&D should be the driver	Manufacturing should be the driver
R&D	3	2
Manufacturing	2	

Not surprisingly, there is disagreement between the organizations. However, the differing opinions in R&D were unexpected.

Issue 3 Lack of processes that secure design for SW download and Car Configuration in production:

This issue was brought up by some of the respondents. They emphasized the need for implementing an "eye of a needle" in the development process where software download and configuration of cars in production specifically were forced to be handled. The participants experienced a risk for not giving priority to these issues since they were managed by informal contacts with electrical manufacturing engineers. Further, it was commented that procedures for securing traceability for car configurations by parameter setting were missing.

Issue 4 Compliance with processes and instructions for quality assured release of software changes in production can be improved:

The company has established well documented processes with instructions and rules for securing that software is quality assured before it is released to production. One of these processes is the software release process described in section 3. However, the participants indicated that the stipulated processes were not always followed. "It is too easy to change software, which leads to deficient quality assurance" was a comment made by one of the interviewees. Contradictory, it was also expressed a need for flexibility to simplify urgent quality issues that could be remedied by rapid software changes.

One major concern was the importance of correct handling of change order documents which are central for the documentation of any product changes. These documents controls what software will be downloaded in the plants and contains information about introducing week, car project, variant etc. Further, they also set the compatibility rules for how software and hardware can be combined in production and after market as well. Some of the interviewees experienced deviating knowledge about change order handling among the developers. Particularly, compatibility between software and hardware was brought up as a frequent problem since competence in this area is sometimes insufficient. The coordination of software changes between projects and running production was also mentioned as a problem.

Issue 5 Prerequisites for appropriate verification of manufacturing processes that are affected by SW in vehicles are not fulfilled:

Verification of the manufacturing processes is a vital phase in product development where the major part constitutes verification on physical test objects during pre series building. A majority of the participants mentioned the problem of insufficient test objects during pre series. A lack of requisite implementation of diagnostics and functional deviations were highlighted as the

main deficiencies. The occurrence of an information gap between R&D and manufacturing for communicating product deviations in pre series was also considered a problem. The combination of defective test objects and insufficient information about the building status makes it difficult to verify the manufacturing processes.

Another consideration that was brought up was the importance of an appropriate feedback system for reporting software related concerns in production.

Issue 6 Lack of possibilities to model manufacturing processes that are affected by SW in vehicles:

Development of scripts for software download and electrical testing constitutes a major part of the preproduction engineering activities for the manufacturing processes that are affected by software in the vehicles. For the time being this work is conducted in the late phases of the product development due to the availability of physical test objects. The general opinion among the interviewees was that modeling manufacturing processes would be supportive during product development. However, the advantages of modeling were not obvious to all participants since some of them had doubts about the potential benefits and how it could be implemented. Specifically, they expressed an uncertainty about how downloading software to physical cars could be replaced by modeling.

Issue 7 There are potential improvements of the manufacturing processes that are affected by SW in vehicles:

The main tasks in the manufacturing processes that are affected by the software in vehicles are downloading software and electrical testing that secure that the vehicles are correctly assembled. The interviewees emphasized three areas of improvement for these processes.

First was mentioned, the need for common processes for downloading software and testing in the production units at the Ford Motor Company (FMC). A lack of communality has an impact on the car projects since the design of software has to be adapted to fit all FMC's plants. One considerable deviation is that software download is not available in all FMC plants. Much effort and a great deal of resources are used to harmonize the design for the different production conditions.

Secondly, most of the interviewees that belonged to the R&D organization expressed their wish to be able to download more software in production since it implies a number of advantages. Rapid handling of quality issues that could be solved with new software and the possibility to reduce the amount hardware variants were particularly mentioned.

The third improvement was expected and deals with the importance of obtaining robust manufacturing processes. Critical factors such as reliable and user-friendly tools, qualified and obliging suppliers of equipment and flawless releases of test scripts were brought up as factors for accomplishing this. The need for automatization and in that way eliminating errors caused by the human factor was also mentioned.

Issue 8 SW competence can be improved within the manufacturing organization:

The increase in software related activities will force the automotive manufacturers to adapt the traditions in mechanical engineering to software engineering. Thus the ability to build up software competencies in the automotive industry will become essential and is presented as one of the challenges in Broy [2].

The general opinion among the participants was that the level of software competence varies in the manufacturing organization. To be able to take the right decisions and make the right priorities in software related issues in production, some of the interviewees emphasized the importance of sufficient software knowledge on the management level. Further, the overall level of software engineering know-how influences the possibility to obtain the right understanding of and attitude towards quality assured handling of electrical faults in the plants.

Another factor that was brought up was better procedures for transferring competence between trained operators used in new model pre series and operators in running production. One aspect named by some interviewees was that there was not dedicated time for educating the running production staff since trained operators are utilized too early for productive tasks. This was reflected by a low First Time Through (FTT) rate for a while after the start of production due to electrical assembly faults.

Issue 9 Knowledge about manufacturing within the R&D organization is important:

Most of the interviewees highlighted the importance of understanding and having insight into the prerequisites for manufacturing vehicles. One of the participants said, "A software designer without knowledge about production of vehicles can cause more damage than usefulness"

The software related manufacturing processes can be divided into two main activities: 1) software download and 2) electrical testing that secures that the vehicles are correctly assembled. Additionally, the interviewees were asked to rate the importance for software designers to have knowledge and understanding about of these activities. The rating was set according to an ordinal scale 1-5, where 1= no importance, 2= less important, 3= important, 4= great importance and 5= very great importance. Table 8 shows the distribution of the responses.

Table 8. Distribution for importance of manufacturing process knowledge

Process	Number of responses for each rating					Total	Median
	1	2	3	4	5		
Software download		3	1	1	6	11	5
Electrical testing			1	2	7	10	5

The responses in table 8 show that a high degree of comprehension of the manufacturing processes among software designers is an essential part of their technical know-how.

Issue 10 There is a need of preserving and facilitating a smooth collaboration between manufacturing and R&D:

The ME role described in section 3 involves electrical preproduction activities by maintaining a close e co-operation with R&D. Most interviewees had good experiences from the collaboration between R&D and ME considering software development. However, they stated that it is based mainly on personal contacts and some interviewees highlighted the necessity of preserving and creating organizational opportunities for proper cooperation. One of the interviewees mentioned that the collaboration between R&D and manufacturing at VCC has a higher prioritization than at the other brands in FMC. Further, a reorganization that was carried out whereby ME and R&D were

separated was considered by one of the participants to be a threat to maintaining a proper co-operation.

5.2 Secondary issues

I11 considers the fulfillment of stipulated routines and instructions for quality assuring development of software. Some of the interviewees experienced differences in the method of working between software developers. The methods were often based on how the one who held the post before and that person's colleagues carried out their work. Further, there were different opinions among the participants considering the knowledge about these processes..

I12 highlights the importance of developing product development systems that are adapted for software engineering. Most of the interviewees mentioned that processes for the development of automotives were mainly designed for developing hardware components. Thus, explicit processes for development of software based systems are not always defined and have to be developed and incorporated into the product development activities.

I13 and ***I14*** consider the ability to smooth the transfer of new technologies to product development projects and allocation of resources in project phases. Some of the interviewees experienced that new technologies introduced in car projects are not mature and have to be developed simultaneously with the new vehicle.

The general opinion among the participants was that most of the resources are spent in late phases of the projects. Some of the interviewees mentioned that 70-80% of the project activities are allocated to the industrialization phase. Although the aim is to concentrate the resources to the early project phases, they claimed that disposable engineering hours must be deployed in high priority assignments in late project phases due to a lack of resources and necessary competencies.

I15 considers decision management which was commented upon some of the participants. One of the interviewees emphasized the importance of making the decision structure clear between the project and line organization for software related issues. Thus, project members are sometimes confused about whether a decision concerned their assignments or not. Further, it was mentioned that the harmonization within FMC increases the complexity of the decision-making process and makes it more difficult for the personnel to take in and implement decision related to software.

I16 to ***I19*** concern requirement engineering. ***I16*** deals with ambiguous product requirements from pre studies in car projects that imply difficulties in interpreting and implementing correct functionalities when they are realized. One of the interviewees thought that this might be caused by not fully understanding and being aware of the possibilities for a new function or changes in the external environment. ***I17*** considers the ability to manage late changes in requirements. To be able to follow the rapid development of software based functions in vehicles, some interviewees highlighted the importance of having agile requirement processes that are adapted for handling late changes in a quality assured way. In ***I19*** the interviewees brought up the need for improving tools and working processes for managing requirements. Most of the interviewees thought that word is not a sufficient tool since it among other things is not a common tool in FMC and does not support reuse of requirements. Further, some participants experienced that requirements are distributed over many locations and it is difficult to find and obtain a survey of all the requirements. ***I18*** deals with validation of requirements. The general opinion among the participants was that inspection and verification of requirements are inadequate. The needs for better test methods and tools and imperative processes for validation of requirement compliance were mentioned by some of the interviewees.

I20 considers modeling of software based functions. The weaknesses of modeling today and its potentials are discussed in Broy [2]. To meet the customer demands for more functions with an increased level of complexity and shorter product cycles, most of the interviewees thought that modeling will become a necessity and has a large potential. However, some of the interviewees experienced that today's tools and stipulated working processes are not sufficient for supporting an efficient and quality assured model based development. This complies with the weaknesses of modeling discussed in Broy [2]. It was also mentioned that a greater involvement of modeling will raise the demands on experience and competence among software developers.

I21 deals with the interaction between VCC and its suppliers of software based systems. Some of the interviewees emphasized the importance of having a close and well functioning relationship with the suppliers since this has an impact on the suppliers' ability to fulfill their assignments. One of the interviewees said that "As it is difficult to write explicit requirements for everything that must be delivered, it is important to have a close communication with the suppliers so that they understand what we want". The sourcing of projects was also commented upon since the price negotiation sometimes resulted in delays in deliveries and unclear commitments. Some participants also highlighted that the suppliers did not have access to the internal database for following up software deliveries.

I22 comprises the co-operation between VCC and other FMC brands and the interaction between functions/disciplines in VCC. The general opinion among the interviewees was that the harmonization with the other brands often causes an increased work load. Moreover, one of the participants experienced difficulties in understanding how the assignments and responsibilities are delegated between the brands. There were different opinions among the interviewees regarding the collaboration between functions/disciplines. Two of the participants claimed that it is sometimes difficult to receive the necessary involvement of other functions/disciplines when software related issues had to be solved together. In contrast, one of the interviewees did not consider this a problem.

I23 considers the general knowledge of SW. A recurrent comment among the participants was that knowledge about SW outside the units that frequently work with SW can be improved in some cases. Specifically, the interviewees highlighted a lack of understanding of the complexity of SW. "They believe that it is only SW and thereby it is easy to change" was mentioned frequently by the participants. This mindset sometimes results in deficient quality assurance of SW changes that are released, which implies a risk for creating other faults.

I24 focuses on the importance of contracting suppliers of software based systems that have the necessary competence and experience in software engineering. According to some of the interviewees this issue has a great influence on the end result. One of the interviewees said that "experienced suppliers simplify the implementation since they already have knowledge about the basic concepts". However, another participant said that it was

often the same suppliers that failed to deliver sufficient quality despite of their experience. Further, deficient processes at suppliers for quality assurance of the software they deliver were also mentioned.

6. Validity evaluation

Four tests to establish the quality of case studies can be discussed in terms of construct, external and internal validity and reliability, according to Yin [3].

6.1 Construct validity

Construct validity concerns establishing correct operational measures for the concepts being studied. One major concern in this study is the impact of one of the researchers' background that was discussed in section 3. The threats can be expressed as respondent biases and researcher biases that were mitigated by utilizing different strategies discussed in Robson [4], see table 9.

Table 9. Strategies used to deal with threats to validity

Strategy	Research bias	Respondent bias
Prolonged involvement	Increases threat	Reduces threat
Triangulation	Reduces threat	Reduces threat
Peer debriefing	Reduces threat	No effect

Prolonged involvement means to learn the culture and to building trust. One of the researchers has worked eight years with electrical manufacturing engineering. This strategy was used to guard against respondent bias.

Triangulation involves the use of multiple sources that enhance the rigor of the research. In this study data were triangulated with interviews, archival records with project information and documents describing processes related to the studied case. Further, observer triangulation was utilized since two researchers conducted the interviews and had the possibility to discuss and analyze the outcome from the interviews.

Peer debriefing means that analysis and conclusions are shared and reviewed by other researchers. This was obtained by conducting the analysis with two researchers and gathering discussion groups where analyses and conclusions were discussed with both research colleagues and members of the studied organization who were familiar with the study. They all agreed with the conclusions of the study.

To evaluate the correctness of the prepared interview guide and analyze the bias from one of the researcher's close relation to colleagues, four pre interviews were conducted.

To maintain a chain of evidence, links between research questions, propositions, issues and raw data on tape and in transcripts were established in the case study database, see table 4

6.2 Internal validity

Internal validity concerns establishing causal relationships, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships according to Yin [3]. For example, the interviewees may not express their real opinions, because they feel restricted by the recording of what they say on paper, and this is a threat to the internal validity. This threat can be limited by participants being guaranteed anonymity by the researchers in interviews. Further, the pre interviews were used to analyze the impact of using tape recorder. The impression was that the interviewees spoke freely and were not disturbed by the recording device. However, this study is primary explorative and according to Yin [3] internal validity is mainly a concern in explanatory case studies.

6.3 External validity

It is not possible to generalize the results of this study to studies at other automotive companies. However, the study was performed at VCC which has a leading role in development of software based systems in FMC that is one of the major automotive manufacturers in the world. This should strengthen the possibility to generalize. The purpose of this study, however, is not to generalize the results since a similar study at another automotive company would most likely result in other findings. Nevertheless, it can be discussed whether some of the findings can appear at similar case studies at other automotive companies and may be generalized. Moreover, the used research method may be applicable to other similar case studies.

6.4 Reliability

According to Yin [3] the objective is to be sure that if a later investigator followed the same procedures as those used by an earlier investigator and conducted the same case study all over again, the later investigator should obtain the same findings and arrive at the same conclusions

To ensure reliability two tactics in case studies are discussed in Yin [3]. The first tactic is to establish a case study protocol with a complete record of all the various activities carried out in connection to the study. This will enable the possibility for another investigator to repeat the case study. The second tactic is to develop a case study database where all the collected data are stored and structured.

To increase the reliability of the results, these tactics have been used in this study. The established case study database contains all data collected together with a complete description of the case study procedures.

7. Discussion of the results

Overall, the results support the proposed theory in this study which anticipated that there are challenges related to the interaction between manufacturing and R&D during development of software based functions in vehicles. The data analysis resulted in 24 issues. These were divided into ten primary and 14 secondary issues mainly depending on their importance to the area of interest in this study and how they were supported by the sources of information.

The accuracy of dividing the findings into primary and secondary issues can be discussed since it is based on the judgment of the researchers involved and members of the organization under study. An extensive investigation may be needed to gain a more solid basis for this division, for example by conducting an inquiry where a number of stakeholders give priority to each issue with regard to their importance to the research area.

Eight of the primary issues in table 5 are supported by both R&D and manufacturing, which indicates concordant views from both organizations for these issues. Contradictory, issue I3 (*lack of processes that secure design for SW download and Car Configuration in production*) and issue I7 (*there are potential improvements of the manufacturing processes that are affected by SW in vehicles*) are only supported by R&D. However, by considering the views from three of the participants from R&D who had experience from manufacturing engineering, also these issues are supported by manufacturing. It can be discussed whether this observation allow to conclude common views also on these two issues. Nevertheless, it gives grounds for such an

assumption, which can be verified by extended interviews with participants from manufacturing,

Table 6 shows that a majority of the secondary issues are only supported by R&D. However, by considering three of the participants' backgrounds as above, all of the secondary issues become supported by manufacturing. A natural explanation might be that these issues have a stronger relationship to the software development at R&D than the primary issues. It can also indicate that there is a lack of insight and knowledge about the software development processes among the participants from manufacturing.

The categorization in figure 3 shows that 12 of the issues discovered had a connection to management & control activities in areas such as project control, competence and collaboration. Although, the other 12 issues were placed in other categories, most of them have some relationship to management. Since this study deals with the interaction between to organizations, it is not a surprise that a majority of the findings can be related to management issues within areas like project control, competence and collaboration.

The results of the study cannot be directly generalized since an inquiry that investigates a similar case in another organization would probable not yield the same findings. In order to enhance the possibility to generalize, it can be discussed whether the issues could be displayed on a higher level of abstraction. However, the risk for presenting too abstract issues which are difficult to link to the research area and not possible to put into practice in the studied organization were considered. Nevertheless, based on the experience from this study, it seems that the research method used is capable of exploring an area of interest and eliciting relevant findings in a structured way. Since the results are valid only for the studied environment they may primarily serve as input for improvements of the activities in the studied case. However, the study may also be useful to other organizations that are interested in founding their development of their work practice on empirically observed findings. To enhance the external validity of the results, more case studies should be conducted on similar contextual conditions.

This study can best be characterized as exploratory and as having as its aim to set the baseline for future research as there is currently a lack of previous empirical case studies in the area of interest. Hence, the report does not provide closer analyses of the results that may explain the findings and be used to investigate possible dependencies between the issues. This could be achieved by extending the study with a questionnaire that contains more structured questions created on the basis of the findings of this study

8. Conclusions

This paper presents an explorative case study of the interaction between R&D and manufacturing focusing on software engineering in the automotive industry. Due to the explorative nature of the study, a qualitative research approach was found most appropriate where data were collected from pertinent documents and in semi-structured interviews. Twelve interviews were conducted with employees at VCC who represent identified key roles that are involved in the interaction between R&D and manufacturing. To enhance the validity of the results, three primary strategies were used: 1) prolonged involvement, 2) triangulation and 3) peer debriefing.

It can be concluded that the results support the proposed theory in this study since the data analysis resulted in 24 issues. These were divided into ten primary and 14 secondary issues depending on their importance to the area of interest in this study.

The results may primary serve as input for improvements of the activities in the studied organization and the study can be useful to other organizations that are interesting in founding their development of work practice on empirically observed findings.

A majority of the discovered issues can be related to management & control activities as this study focuses on the interaction between two organizations.

9. ACKNOWLEDGMENTS

The authors would like to express their gratitude to all those somehow involved in this study and especially the interviewees at VCC.

10. REFERENCES

[1] Klaus Grimm, Software Technology in an Automotive Company – Major Challenges, Proceedings of the 25th International Conference on Software Engineering, IEEE 2003.

[2] Challenges in Automotive Software Engineering, Broy, Manfred (Institut fur Informatik, Technische Universitat Munchen), Proceeding of the 28th International Conference on Software Engineering 2006, ICSE '06, 2006, p 33-42

[3] Yin R (2003) Case study research: design and methods. 3rd edn. SAGE Publications, Inc., Thousand Oaks,

[4] Robson, C., Real World Research: A Resource for Social Scientists and Practitioners-Researchers, ,Second edition, Blackwell, 2002.

[5] Experiences from conducting semi-structured interviews in empirical software engineering research, Hove, S.E.; Anda, B.;Software Metrics, 2005. 11th IEEE International Symposium 19-22 Sept. 2005 Page(s):10 pp.

[6] B.G. Glaser and A.L. Strauss, The Discovery of Grounded Theory:Strategies for Qualitative Research. Aldine Publishing, 1967.

[7] M.B. Miles and A.M. Huberman, Qualitative Data Analysis: AnExpanded Sourcebook, second ed. Thousand Oaks, Calif.: Sage, 1994.

[8] A case study investigating the characteristics of verification and validation activities in the software development process; Berling, T.; Host, M.;Euromicro Conference, 2003. Proceedings. 29th 1-6 Sept. 2003 Page(s):405 –

[9] Engineering Automotive Software
Broy, M.; Kruger, I.H.; Pretschner, A.; Salzmann, C.;
Proceedings of the IEEE
Volume 95, Issue 2, Feb. 2007 Page(s):356 – 373

[10] Jönsson, P. and Wohlin.C (2005b): Understanding impact analysis: an empirical study to capture knowledge on different organisational levels. In proceedings of the International Conference on Software Engineering and Knowledge (SEKE05), pp. 707-712, July 2005, Taipei Taiwan

An Empirical Evaluation of Domain-Specific Language Tools in the Context of Service-Oriented Architectures

Ola Lindberg, Peter Thorin, Miroslaw Staron

IT University of Göteborg
Gothenburg, Sweden
d02ola@ituniv.se, thorinp@ituniv.se, miroslaw.staron@ituniv.se

ABSTRACT

The new initiative of Microsoft – Software Factories – is gaining popularity as it is often perceived as an alternative realization (to Model Driven Architecture) of the vision of model-driven development. In this paper we evaluate Software Factories from a perspective of their usability in software projects. In particular we focus on the effort required to develop a specific software factory and then compare a developed solution to the existing ones developed using UML-based technology. The results show that Software Factories further improve the practice of software development by decreasing the time to design a software factory and by decreasing the effort required to develop software using the developed factory. The study is done in the context of modeling of web services.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design tools and techniques – *Computer Aided Software Engineering (CASE), Object-oriented design methods.*

General Terms

Design, Languages.

Keywords

Domain Specific Modeling, UML, case study

1 INTRODUCTION

The principles of Service-Oriented Architectures (SOA) make it possible to create loosely coupled applications in dynamically changed environments which supports flexibility in applications with changing requirements [1]. One of the intended benefits of using SOA is the shift from development of proprietary components to creating networked, pluggable, and reusable units of functionality. This shift to networked and dynamic environments requires new methods for developing software. In particular, the development companies need to be able to quickly customize their services or produce new ones which fulfill new requirements. The services are required to operate on various platforms and integrate various technologies, which in consequence require high portability.

Software Factories [1] seem to be one of the potential solutions to the above demands. The main idea behind software factories is that there exist a set of integrated domain-specific modeling languages (DSLs) which allow modeling a particular part of software and generating full source code for that part. By exchanging the generators the users of software factories can rapidly adapt their services for other environments or produce services with extended functionality.

An alternative approach is to model services using the Unified Modeling Language (UML, [2]) and utilize its extension mechanism – profiles. This approach is the core of the OMG's initiative – Model Driven Architecture [3, 4]. In the course of this paper we intend to use a pre-defined UML profile to model a reference system and then repeat it using a created software factory. The case study presented in this paper is aimed at evaluating how usable software factories (in particular its main component – DSLs) are from the perspective of their users. Therefore we address the following research question:

> *How usable are software factories from the perspective of their users during software development?*

In this case we focus on *line* software development which we define as the development of core business systems in the company. In particular, it is not our intention to evaluate software factories from the perspective of language engineering (c.f. [5]). Nevertheless we analyze the time required to create a simple language in order to provide initial data for performing the cost-benefit analysis of using software factories.

The results of our case study show that software factories decrease the time required to develop these core business systems while at the same time decrease the time required to develop the software factory. This is compared with the baseline development using UML profiles.

The paper is structured as follows: Section 2 presents the most related work in the area. Section 3 presents the concept of software factories. Section 4 presents the design of the study and Section 5 presents the results from the study. Section 6 contains conclusions.

2 Related Work

Vokáč and Glattere [6] have created a UML-based DSL which is specifically targeted at SOA. It consists of a UML Profile to provide the domain-specific elements in UML models. They have also created a generic code generator for generating code from profiled models. The authors of that paper did not use software factories and their experiences show that it took a senior developer two years to develop the UML Profile, the code generator and associated tools [6 pp. 500, 502].

Wada and Suzuki [7] developed a complete Model-Driven Development Framework for SOA called mTurnpike. In this framework, a DSL is defined as a "metamodel that extends the UML 2.0 standard (superstructure) metamodel with UML's extension mechanism" (UML Profiles). mTurnpike has separated the two tasks of creating Domain-Specific Models (DSM) and Domain-Specific Code (DSC) so that modelers and programmers do not need to know how Domain-Specific Concepts are implemented and deployed in detail ([7], pp. 587-588). Even though the mTurnpike is not specifically targeted at SOA, Wada and Suzuki [7] have created an example DSL for modeling SOA. In software factories both DSM and DSC are used as part of a single factory.

Staron and Wohlin [8] conducted a case study at Telelogic AB in Malmö, Sweden. The goal of the study was to elaborate industrial criteria for choosing between UML Profiles and UML metamodel extension (which are used as DSLs in that context). The problem is that modelers at small and medium companies needs help to decide whether to use UML Profiles, which are relatively cheap, or the more costly (and more powerful) UML metamodel extensions in order to provide required customization of the language, which effectively means creating a DSL. The case study resulted in nine different industrial criteria, where three where of a business nature and the remaining six are of a technical nature. However, the most interesting part of the case study with relation to this study is the Section on effort and resources [8], which states that it takes approximately 3-4 weeks for several engineers to extend the UML metamodel, whereas it takes approximately 1 week to create a UML Profile for a single developer. In this paper we intend to contrast that result with the results of our case study.

3 Software Factories

The initiative of Software Factories [1] originated as a means of using DSLs in order to improve productivity and quality of software product lines. The overview of software factories is presented in Fig. 1.

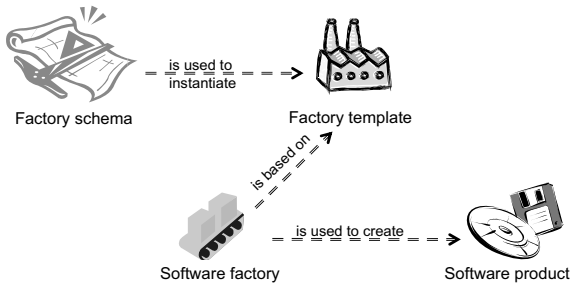


Fig. 1. Overview of software factories

In order to develop a set of software products in a short time the developers use a software factory during their software development tasks. The factory is based on a factory template which describes how a particular factory is created. The factory template is an instance of a factory schema, which describes what kind of elements can exist in software factories and what the dependencies between them are.

The core of software factories is the DSL technology – i.e. a technology for creating and using small, dedicated DSLs in the course of software development. The structure of software factories in the .NET technology is presented in Fig. 2.

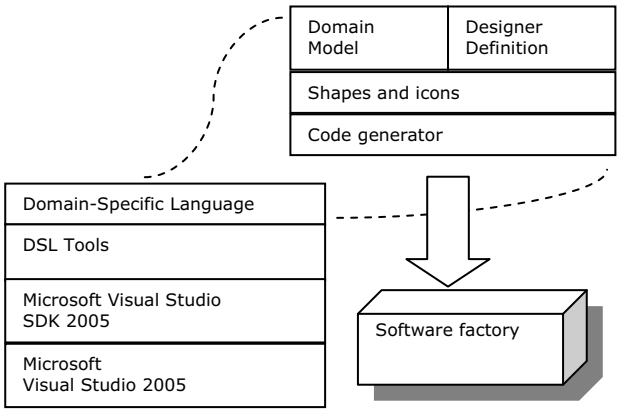


Fig. 2. Components involved when using DSL Tools

A complete software factory contains a created DSL and a set of associated elements built into the Microsoft Visual Studio 2005 SDK. It is a unit of deployment for developers who use the factory to create the core business assets for their companies. Domain model and Designer definition constitute the abstract syntax of the DSL, shapes and icons constitute the concrete syntax and the code generator defines translational semantics of the language (c.f. [9]).

In this paper we focus only on evaluation of creation of software factories – i.e. instantiation of a factory template and software development using a particular factory. In the case of our paper we use an example factory for creating web service based application. In this research we do not consider the creation of factory schemas therefore, we have the following elements:

- Software product – an example system – driving direction web service application
- Software factory – web service software factory

The web service software factory contains a DSL and a code generator to WSDL (Web Services Definition Language, [10]). It is developed as part of the case study presented in this paper. The details are presented in Section 4.

4 UML Profiles

UML contains a definition of stereotypes, and specifies them as one of the possible extension mechanisms of the language. In UML, the stereotypes are a way of adding a new semantics to the existing model elements. They allow branding the existing model elements with new semantics, thus enabling them to "look" and "behave" as virtual instances of new model elements [8, 9]. During the evolution of UML (from version 1.1 [8] to 1.5 [9]), the definition of stereotypes in the UML metamodel has not changed significantly, although underwent minor revisions due to the changes in the definition of other extension mechanisms (mostly *tagged values*, which evolved from being merely additional information for code generators in UML 1.1 specification towards virtual links between metamodel elements in UML 1.3 and later). With a growing UML tool support for this mechanism (as opposed to the lack of support for metamodeling) the stereotypes are beginning to play a major role as a means of visualizing the provision of UML as a family of languages rather than a one-fits-all modeling language.

The notion of profile appeared for the first time in the UML specification version 1.3. Since its introduction, the profiles provide a way of organizing UML extension mechanisms to some extent. Firstly, they provide a means for defining new languages that are based on UML. A set of stereotypes which are mutually dependent and connected (which is expressed in a set of constraints attached to stereotypes definitions) can be closed under a specific profile. Such a profile forms a complete whole and enables using UML as some other notation (for example the Entity Relationship Diagrams Profile allows to use UML class diagrams as Entity Relationship Diagrams). Examples of such profiles are the UML Data Modeling Profile [12] and the Entity Relationship Diagrams Profile [13]. Secondly, the profiles allow to group elements which are not tightly interrelated, thus providing libraries of modeling elements. Such profiles do not define constraints on using the elements, i.e. there are no constraints attached to stereotypes. An example of such profile is the UML Profile for Business Modeling [9].

5 Evaluation method

We chose to perform an experimental development of a simple language for modeling interaction between web services – i.e. software based on service oriented architecture. The requirements for the language – the elements which needed to be included in the language were identified by prioritizing a set of elements found in literature describing SOA. The results of the prioritization (elements in the language) are presented in Table 1. The priority is the number of literature sources where the element was used. The higher the priority the more important the element is.

All elements with priority larger than 4 are included into the DSL for SOA – herein referred to as SOLA. From these prioritized elements, a requirement specification for SOLA was created. Some of the elements, even though they had different names, were duplicates. Since there was no standard definition for SOA that could be used as a guideline, these duplicates had to be grouped into unique elements. The grouping was done by analyzing the context of the sources.

Table 1. Language elements for SOA

Element	Literature Sources	Priority
Service	[7, 12-19]	9
Service Interface	[7, 11-17, 19]	9
Message	[7, 11-17]	8
Service Description	[7, 12-16, 19]	7
Service Client	[12, 14, 16-19]	6
Service Repository	[12-16, 19]	6
Service Aggregator	[7, 12, 13, 17, 19]	5
Service Provider	[12, 16-18]	4
Service Data	[11, 14, 16]	3
Service Operation	[12, 14, 17]	3
Service Logic	[14, 16]	2
Service Connector	[7, 17]	2
Service Protocol	[12, 17]	2
Message Attachment	[17]	1
Port	[13]	1
Service Bus	[14]	1
Service Gateway	[17]	1
Service Partition	[17]	1
Service Stub	[14]	1

The data collection was made by Lindberg and Thorin as a part of their Bachelor Thesis. They had both worked with UML in several student projects and attended UML courses, but neither had any previous experience of DSL or SOA development. The rationale behind using the number of references as a prioritization technique was that we wanted to identify the most commobly referred elements in the literature – the more authors discuss a particular element, the more important it is in the SOA domain.

5.1 Data Collection

During the study we collect the following data:

- Effort required to learn the new technology (person-hours[1])

  In order to assess the effort required to understand and start using the new technology we counted the hours needed to start using software factories and modeling service oriented software.

- Effort required to develop a modeling language (person-hours)

  To evaluate the effort of using SOLA, it was compared to using a UML 2.0 Profile for Software Services [17] developed by IBM, further referred to as the IBM Profile.

  Tasks are timed using Work Timer. In order to compare that to the IBM Profile we use the data from an industrial case study at Telelogic by Staron and Wohlin's [8].

- Effort required to model the example system using both SOLA and the IBM Profile (person-hours)

  An example system was modeled in both IBM Profile and in SOLA. The modeling was timed in order to facilitate comparison. Modeling is defined as the task of creating a model according to a specification, as opposed to the creation of a specification of a model.

[1] We shall round the total time to full person-hours.

- Expressiveness of the language
  - Number of available concepts

    The number of elements which are available in the language (e.g. Service, Service Provider).
  - Number of available relationships

    The number of relationships between elements (e.g. Service consumption).
  - Number of available properties

    The number of properties of each language element.
- Number of elements used to model the same software

  The number of elements of the language instantiated to model the example system.

The time was measured by task and individually for each participant. Pair-development was utilized for all of the tasks with the exception of literature reviews. At the end of the study the time spent on each task was summed and the individual times for each participant were combined to person-hours.

Vokáč and Glattere encountered the problem that screen area was soon consumed due to how their UML Profile was implemented, which in itself was due to how UML is implemented [11]. The problem was that one cannot create relationships between the properties of a concept, and therefore they created concepts for these properties, which were linked to the main concept with a relationship. This meant that the properties could be reused, but also that the number of visible elements on the screen increased significantly since more concepts had to be visible. The only solution to the problem they could see was to use a tool that was built to handle DSL.

5.2 Comparison of usability of SOLA and the IBM Profile

The IBM Profile was used in IBM-Rational Software Modeler and SOLA was used in MS Visual Studio 2006. The modeling of the example SOA was split into three different increments, to avoid or minimize that all learning was done in only one of the tools and therefore affect the result when using the other tool. In the first increment, the first tool to be used was SOLA (Fig. 3) and the second tool was IBM RSM. This changed in increment 2 so that IBM RSM was used before SOLA.

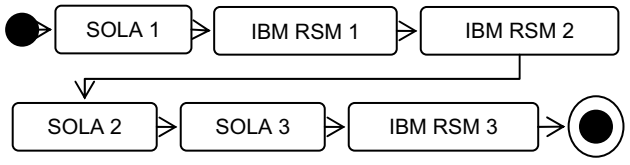


Fig. 3. Incremental process used while modeling

The completeness of the model increased throughout the increments, i.e. increment 3 started with the result of increment 2, and increment 2 started with the result of increment 1. The models were not recreated from the beginning in each increment.

6 Results

Section 6.1 presents the data collected when developing SOLA and developing the example system, as described in Section 4.

6.1 Development of SOLA

The development of SOLA required the execution of a number of tasks, both theoretical and practical. The first tasks were theoretical and entailed learning several theories and models, such as SOA principles and DSL engineering. The effort required to learn the associated technologies is presented in the chart in Fig. 4. The literature review performed to define a SOA took 32 person-hours. A literature review on DSL was also conducted and DSL Tools had to be learnt. The literature review took 11 person-hours and learning DSL Tools took 28 person-hours. In total, the learning effort was 72 person hours.

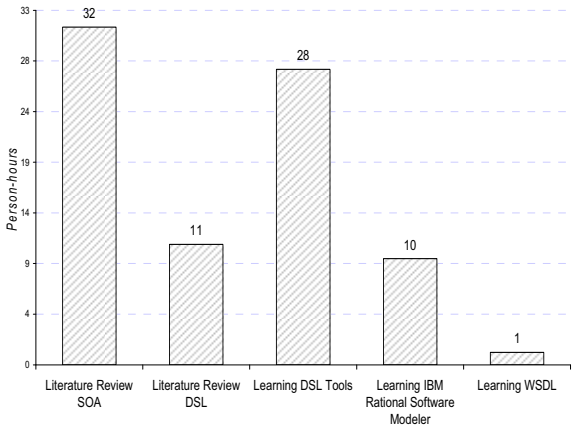


Fig. 4. Person-hours assigned to learning tasks

In comparison to the 28 person-hours spent on learning the DSL Tools, it took only 10 person-hours to learn IBM RSM to the same extent. The learning outcome of this process was to be able to develop a simple DSL (or a profile in case of IBM RSM) and to be able to use the DSL (or the profile) for modeling a simple system (but not the system which is used in this case study). It seems that using UML Profiles requires less training effort, which is understandable since the DSL technology is new and contains a number of new concepts compared to UML Profiles. Fig. 5 shows the effort in person-hours for developing abstract and concrete syntax, and the code generator for WSDL.

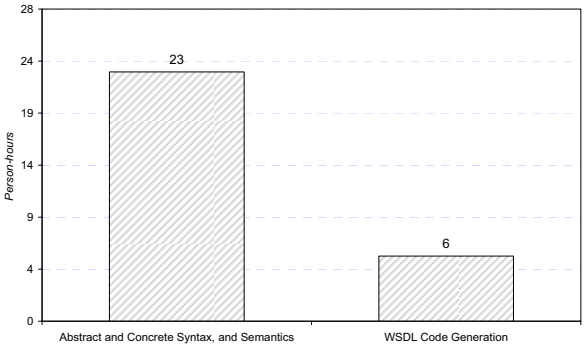


Fig. 5. Person-hours required to develop SOLA

In comparison to that we use the data from a case study performed in 2005 (i.e. at a similar technology maturity to the current situation) at Telelogic by Staron and Wohlin [8] which contains the effort required to develop comparable

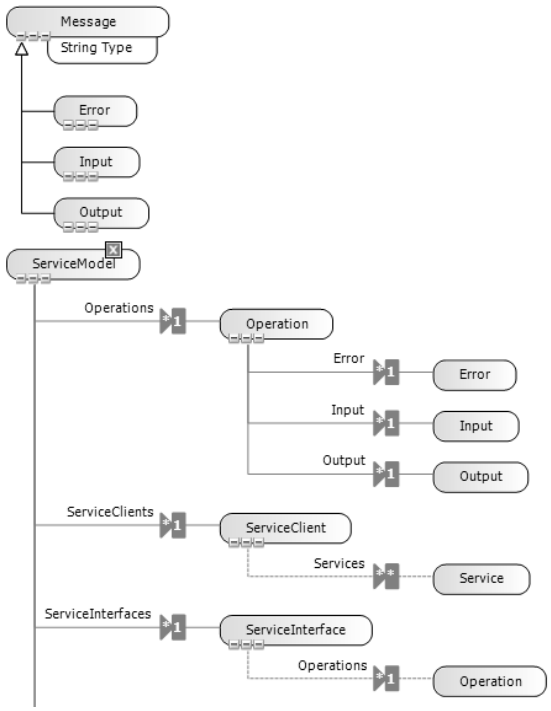


Fig. 7. Excerpt from the definition of SOLA

(w.r.t. functionality and size) metamodels and profiles. The comparison is presented in Fig. 6.

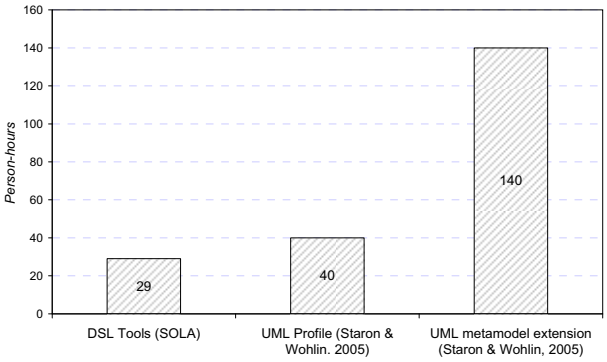


Fig. 6. Person-hours required to develop DSL

The effort required to create SOLA, 29 person-hours, is comparable with that it takes approximately 40 person-hours to develop a UML Profile and approximately 140 person-hours to extend the UML metamodel [8] (c.f. Fig. 6.). In order to understand these numbers, it is important that we realize what the different approaches entail. The excerpt of the resulting definition of SOLA is presented in Fig. 7. When using DSL Tools, it is possible to define the characteristic for each element without the limitations which the mechanism of UML Profiles sets on the language. If the limitations set by choosing a UML Profile are acceptable, the effort to create SOLA and the UML Profile created in [8] is somewhat similar. But if the needs exceed the possibilities provided by UML Profiles and an extension of the UML metamodel must be implemented the effort required to create the DSL in UML increases dramatically. Instead of the one week required to create the UML Profiles, one might have to assign several engineers for approximately 3-4 weeks, as in the case studied by Staron and Wohlin [8].

6.1.1 Language Size

A basic characteristic of both SOLA and IBM Profile for modeling web services is presented in Table 2.

Table 2. The number of available elements in SOLA and the IBM Profile

	SOLA		The IBM Profile	
	Concrete	**Abstract**	**Concrete**	**Abstract**
Relationships	8	16	1	13
Properties	20	18	18	19
Concepts	7	12	8	12
Total Elements	35	46	27	44

The number of elements for the abstract syntax in SOLA (46) was counted from the definition of the abstract syntax in DSL designer. The number of elements for the abstract syntax in the IBM Profile (44) was counted on an UML 2.0 metamodel of the profile included in the documentation of IBM RSM [17]. The names of the relationships in the IBM Profile metamodel were not counted as properties, since the SOLA metamodel has no comparable names.

As it is presented in Fig. 8, the number of elements for the concrete syntax in SOLA (35) was counted in the specification for SOLA. The number of elements for the concrete syntax in the IBM Profile (27) was counted in the specification for the IBM Profile.

The results show that SOLA contains more elements than the UML Profile for modeling web services. The functionality of both SOLA and this UML Profile are almost the same, with small exceptions regarding graphical notation, which are dictated by the underlying technology (DSL vs. UML).

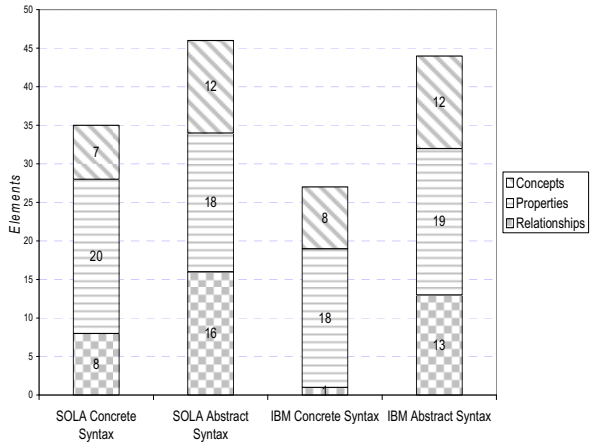


Fig. 8. Number of available elements in SOLA and the IBM Profile

6.2 Developing an example system using SOLA and the UML Profile

The modeling of the example system was divided in incremental parts in order to avoid so called bias effect, as described in Fig. 3. The system was a system for creating driving directions through a set of web services.

Not all elements in the IBM Profile exist in SOLA. This is due to the fact that those elements were not highly prioritized when the requirement specification was created. This may affect the result of the measuring of the available elements. In the increments 1 and 2 it was possible to model the same elements in both tools, but in increment 3, the Service Repository was not available in the IBM Profile; as it is presented in Fig. 9.

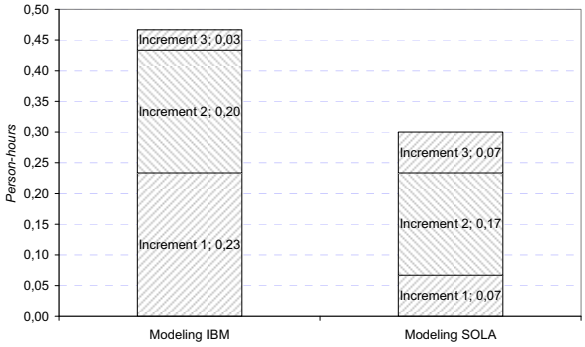


Fig. 9. Person-hours spent on modeling example SOA

It took a total of 0.3 person-hours to model all increments in SOLA, and 0.47 person-hours in the IBM Profile, which can be seen in Fig. 9. This means that there is a difference of 0.17 person-hours when modeling SOA using SOLA and the IBM Profile. However, this small amount constitutes more than 35% percent of the modeling time, and that indicates a significant difference when a real system would be implemented and the required time is more than half a person-hour.

The number of elements used to model the example system is presented in Table 3.

Table 3. Number of elements used to model the example system

	SOLA	The IBM Profile
Relationships	16	5
Properties	38	40
Concepts	15	9
Total Elements	69	54

SOLA has a larger amount of elements, 69 vs. 54. The main differences are in the number of relationships and concepts. The reason for this is because in the IBM Profile several relationships are not shown as lines in the model but as embedded or hidden properties of the concepts.

7 Conclusions and Future Work

In this paper we evaluated software factories from the perspective of the users of this technology. The focus of this evaluation was to compare using DSLs with using UML Profiles which can be seen as an alternative to DSLs. The evaluation was done by comparing two parts – building an example DSL (or a UML Profile) and using the DSL (or the UML Profile) to develop an example system.

The results from the evaluation show that the effort of development of a DSL can be compared to the effort of development of a UML Profile. However, if the needs exceed the possibilities provided by UML Profiles and an extension of the UML metamodel must be implemented the effort required to create a DSL is approximately three times shorter (less than one person week to develop a DSL compared to 3.5 person weeks to develop a metamodel extension). Using the DSL seems to lead to shorter development times than using the UML Profile. However, the example system was rather small and the difference was negligible. Our further work is focused on further evaluation of DSLs in an industrial context to obtain more detailed data on the effort required to develop systems using DSLs.

The expressiveness of the DSL created in the case study, however, was much lower than the expressiveness of the UML Profile. More DSL elements were required to model the same system than what would have been required when using elements from the UML Profile. This could be caused by the fact that in our comparison we used an existing UML Profile which was developed by a company with significant experience in this field.

In conclusion the case study presented in this paper indicates that the DSL technology from Microsoft is a promising alternative to model driven software development based on UML and its extension mechanisms. However, further evaluations are needed since the differences between these two technologies are not as eminent as one could have expected.

References

1. Greenfield, J., Short, K.: Software factories : assembling applications with patterns, models, frameworks, and tools. Wiley Pub., Indianapolis, IN (2004)
2. Object Management Group: Unified Modeling Language Specification: Infrastructure version 2.0. Vol. 2004. Object Management Group (2004)
3. Mellor, S.J., Kendall, S., Uhl, A., Weise, D.: Model-Driven Architecture. In: Bellahsene, Z. (ed.): Object-Oriented Information Systems, Vol. 2426. Springer-Verlag, Montpellier (2002) 290-307
4. Miller, J., Mukerji, J.: MDA Guide. Vol. 2004. Object Management Group (2003)
5. Evans, A., Maskeri, G., Sammut, P., Willians, J.S.: Building Families of Languages for Model-Driven System Development. Workshop in Software Model Engineering, San Francisco, CA (2003) Np
6. Vokac, M., Glattetre, J.M.: Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service Oriented Application - Experiences and Challenges. In: Briand, L., Williams, C. (eds.): Model Driven Engineering Languages and Systems, Vol. 3713. Springer-Verlag, Montego Bay, Jamaica (2005) 492-506
7. Wada, H., Suzuki, J.: Modeling Turnpike Frontend System: A Model-Driven Development Framework Leveraging UML Metamodeling and Attribute-Oriented Programming. Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005. Springer Verlag, Montego Bay, Jamaica (2005) 584-600
8. Staroń, M., Wohlin, C.: An Industrial Case Study on the Choice between Language Customization Mechanisms. Improving Modeling with UML by Stereotype-based Language Customization, Vol. 2005:08. Blekinge Institute of Technology, Karlskrona (2005) 95-126
9. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied Metamodeling - A Foundation for Language Driven Development. Xactium (2004)
10. Bardram, J.E., Christensen, H.B.r., Corry, A.V., Hansen, K.M., Ingstrup, M.: Exploring Quality Attributes Using Architectural Prototyping (2005)
11. Vokáč, M., Glattere, J.M.: Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service-Oriented Application — Experiences and Challenges. Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005. Springer Verlag, Montego Bay, Jamaica (2005) 492-506
12. Papazoglou, M.P., Georgakopoulos, D.: Service-Oriented Computing: Introduction. Commun. ACM **46** (2003) 24-28

13. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in Web services. Commun. ACM **46** (2003) 29-34
14. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA. Pearson Education, Inc., Upper Saddle River, NJ (2005)
15. Fabio, C., Eric, S., Umeshwar, D., Ming-Chien, S.: Business-oriented management of Web services. Commun. ACM **46** (2003) 55-60
16. Jørstad, I., Dustdar, S., Thanh, D.V.: A Service-Oriented Architecture Framework for Collaborative Services. 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise. IEEE (2005)
17. Johnston, S.: UML 2.0 Profile for Software Services. Vol. 2006. IBM developerWorks (2005)
18. Littlewood, B., Fenton, N.E., City University Centre for Software Reliability: Software reliability and metrics. Elsevier Applied Science, London (1991)
19. Thanh, D.v., Jørstad, I.: A Service-Oriented Architecture Framework for Mobile Services. Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop. IEEE (2005) 65-70

Evaluation of Automated Design Testing using Alloy

Johannes Andersson
Lund University
Dept. Computer Science
Box 118, SE-221 00 Lund
+46 76 136 16 38
johannesa@gmail.com

Per Runeson
Lund University
Dept. Computer Science
Box 118, SE-221 00 Lund
+46 46 222 93 25
per.runeson@cs.lth.se

ABSTRACT
An abundance of tools and notations are available to help software designers in their work, but few of these can provide any form of analysis of a design or verify that desired properties are upheld. During the last decade, however, some progress towards machine assisted verification of design has been made. In this paper, a combined design notation and analysis tool called Alloy is evaluated. Alloy is used to study an existing piece of software in development at a telecommunications company and is then evaluated according to how useful it can be from a perspective of cost versus benefit in an industry context. Both the process of learning and applying Alloy is studied. It is concluded that it can be quite time consuming, and thus costly, to learn Alloy unassisted. It is also concluded that the benefits when studying a pre-existing design with Alloy are sharply constrained by that design. That is, if there are few clear design abstractions to test, and a redesign is not made, or the scope of the study changed, then the benefit of using Alloy is found to be minimal.

Categories and Subject Descriptors
D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification

General Terms
Design, Economics, Reliability, Languages, Verification.

Keywords
Alloy, industry context, practical results, evaluation

1. INTRODUCTION
In software development, finding the right design abstractions can have substantial benefits for an implementation in code. It is likely that code resulting from a clean and coherent design will gain some benefits compared to a deficient design in at least the following areas: reliability, simplicity, maintainability, efficiency and correctness. Of course it is a given that every development process strives to achieve the goal of a good design for its end product, be it a strictly design driven process such as the waterfall method or an iterative process such as extreme programming [3]. But while there may be many notations and theories that can assist in creating a good design, there are few tools that let a user verify and simulate a design without implementing it outside the design environment itself.

A powerful design tool may be advantageous because it would allow verification and correction of design properties in a system model. A smaller automatically verifiable system model may be more effective for detection of systematic defects than test and analysis of a full implementation. However, two things will determine the practical usefulness and success of such a tool. The first, how much overhead is incurred in application of the tool, and the second, how much practical benefit can be observed by applying the tool.

This paper presents an evaluation of a design notation and associated analysis tool called Alloy [6, 7]. Alloy has been developed by Daniel Jackson and his research team of the software development group at MIT. According to Jackson, Alloy has been specifically designed to model and analyze software abstractions, and it has been successfully used in a number of instances, for example, to explore file system synchronization policies [1], to formally specify a military security protocol [8], and to model processes [10].

In this article Alloy is used to study the design of a software component in a Push-to-talk application for mobile phones, developed at Teleca Mobile in Sweden. The Push-to-talk application itself is part of greater framework developed at Teleca Mobile called IMS, IP Multimedia Subsystem, which implements media delivery to mobile phones based on Internet connections.

Alloy is evaluated based on the results from the study of the Push-to talk application. Not only are the direct results of using Alloy, such as possible improvements identified and problems discovered, considered in the evaluation, but also the cost in terms of time used to achieve those results. For an industry context, it is at least as important to know how much something costs to do as what it can achieve.

The evaluation was made as a part of a master's thesis by one of the authors (Johannes). His background is in computer science engineering with a focus on software development, and he was a novice at using Alloy and similar tools at the start of the master's thesis. This means that the results of the evaluation may

Submitted to SERPS07

be indicative of ease or difficulty in learning Alloy as well as the effects of using it.

2. MOTIVATION AND OBJECTIVE

In the literature study leading up to this paper[1], no attempts to evaluate Alloy from the context of every day use in an industry context were found. The main goal of the evaluation described in this paper is to determine the practical value of Alloy use in day-to-day work with software development in an industrial setting. A consequence of this is that there is no attempt to find a "perfect problem" to evaluate Alloy with. Instead Teleca is asked to provide a part of a design that has exhibited some problems that they wanted analyzed and the resulting evaluation is limited to this requested area. The practical needs of the industry context are thus prioritized over the needs of the study in generality and risk minimization. This means that the study by itself may lack some validity, but should give a truer picture of actual practical usability of the tool in a context with other studies of a similar nature.

The practical context also influences other decisions made about study. The target group for Alloy users is "ordinary" developers. I.e. software engineers looking for new methods to improve the software development process, but without any institutionalized support in the form of tutors or significant academic schooling in the area of the new method (in this case Alloy). This obviously affects the efficiency of any method, especially one as different from basic software development as Alloy, but it is also part of the objective to ascertain if Alloy can retain some efficiency without a large pre-existing support structure. Easy adaptation for a developer previously untrained in formal methods would be a major advantage for Alloy because it cannot be assumed that a very large section of the active workforce in the software industry today has relevant training.

3. BACKGROUND

3.1 Automated Verification Methods

In this paper, an "automated verification method" is a method that can be used to verify specified correctness properties automatically for a design model. That is, a method that will find whether a property is true or false, for a given model of a design, without human intervention. This is in contrast to interactive methods that can be used to verify correctness properties but require input from a skilled human operator during the verification process. More about the difference between interactive and automatic methods can be found in the introduction to Clarke-Schlingloff [2].

3.2 Properties of Automatic Verification Methods

Alloy is one tool that can be used to perform automatic verification of design models today. There are a number of other tools available, for example SPIN [4] and UPPAAL [5]. While Alloy and the other tools each are different in many specifics, all encountered in this study share two basic properties. First they all make use some form of specification language. This language must be used to create a model of the design to be verified, and to specify what properties to verify. Second, the way the tools "prove" a concept true or false is by using a form of brute force approach. That is, they test that a given property holds for all possible states that a model can attain within a limited scope. Exactly what scope and limited means varies with the method of test being used. The general meaning of scope however, is that the size of the problem must be kept limited in some way, so that the total number of states to be tested also remains limited.

In the literature, two underlying analysis methods that enable automatic model verification have been identified. These methods are *model checking* and *SAT* (Boolean satisfiability) *solving*. Model checking, in short, performs a search through a graph of state transitions and checks whether a requested property holds for all transitions. For more details on the mechanics of model checking, see Clarke-Schlingloff [2]. SAT solving requires that the state of the model be converted to a single boolean expression. A SAT solver will then be used to attempt to find an instance where the expression is true.

Both model-checking and SAT-solving explore a space of states whose properties are derived from logical formulas. According to Jackson [6], a model checking tool was an inspiration for his tool Alloy, but he found model checking insufficient for analyzing software models. Instead Alloy uses SAT solving to analyze its models.

It has also been shown that SAT tools can be used to improve the efficiency of model-checking [11]. The distinction between SAT solving and model checking in terms of what the effects of choosing a particular analysis method for a particular problem is probably worthy of a study of its own. However, since this paper is primarily concerned with evaluating the practical usability of Alloy, a more in-depth study of the technical properties of these methods will not be pursued here.

As for usability, the main benefit of these methods is of course the power of the automated verification. There are some limitations to overcome to be able to utilize this power. One of these limitations, the difficulty of constructing a model and finding relevant properties to verify, are studied in this paper. Another limitation is that since these methods utilize a brute-force approach, there is also an inherent limitation in the complexity of problems that can be verified. A case where this limit was encountered is described by Hashii [8] where the model of a security protocol for military aircraft had to be simplified to allow verification with the Alloy analyzer.

The results of the analysis with an automated method also deserve some attention. Since the analysis is focused on finding faults rather than proving correctness, a method will find a faulty state of the model if one exists within the scope of the analysis. The faulty state found is called a *counterexample* and can be quite useful for finding an error. In the simplest form, a counterexample simply shows a state that can occur in a system with a specific design. However, programs can allow a more complex trace to show for example how an error is reached from a specific, correct, starting state. Two programs with this type of functionality are UPPAAL [5] and Alloy [6].

Automated verification methods have been used in a variety of applications, and are not limited to evaluating explicit

[1] Performed in December 2006

software or hardware designs. As long as the target application can be modeled in an appropriate language, automatic verification may be used to test properties. For example; the design and analysis of a gear controller by Lindahl [12] and modeling the process of a student going through an education program by Wallace [10].

4. ALLOY

The Alloy Analyzer is a tool developed by the Software Design Group [at MIT] *for analyzing models written in Alloy, a simple structural modeling language based on first-order logic. The tool can generate instances of invariants, simulate the execution of operations (even those defined implicitly), and check user-specified properties of a model.* [7]:

In the introduction to his book on Alloy [6], Jackson elaborates on the importance of the abstractions that software is built upon. He claims that flaws in design are not discovered sooner because traditionally a designer does not have access to any automated aids to verify the correctness of his/her assumptions. A programmer, on the other hand, will at least have a compiler and test cases to help him/her spot most ambiguities. Jackson argues that code is a bad medium to explore design issues however, since code contains a lot of implementation related details that obscure the underlying design with an extra layer of complexity.

Alloy has been designed to give software developers the power to explore design issues and model abstractions in a more efficient way than has previously been possible. Its main goal is to let its users explore and extensively test abstractions of a design before actually committing any of it to implementation level code.

The Alloy language is based on first-order logic and set theory and has outward (and some functional) similarities with object oriented languages such as Java and C++. Its main building blocks are atoms and relations. Because an atom is a primitive entity that is indivisible, immutable and uninterpreted, relations are used to capture the properties that atoms in themselves cannot represent, for example ordering, structure and properties of items. To make the language first order and analyzable, there are some limitations to what can be expressed. An example is that relations can map atoms to other atoms, but not relations to other relations.

Design testing in Alloy can be roughly divided into three steps. Firstly, a model is created in the Alloy language, which is checked for syntactic and type consistency. Secondly, this model is transformed by the Alloy Analyzer into an expression that an SAT solver can understand and analyze. Thirdly, if the SAT solver can find a solution, this is sent back to Alloy, which will provide a visual representation of the found solution, or report a negative if no solution was found. That the SAT solver finds a solution can be desired or not depending on whether you are looking for an example of execution or testing for an error, but the basic mechanics are the same.

5. EVALUATION PLAN

The purpose of the evaluation is to examine how much time and effort is needed to employ Alloy in practice in a project being developed at a telecommunications company. The evaluation is also intended to give an overview of what benefits are gained, and what problems and pitfalls are encountered.

5.1 Evaluation Method

This evaluation is based on the concept of action research as described in Höst et al. [9]. The concept of action research can be roughly described by four steps:

- Plan – Identify a problem and its causes
- Do – Propose and apply actions to solve the problem
- Evaluate – Check that suggested actions led to improvements
- Learn – Make use of the knowledge gained in the study to suggest and/or decide future action

The evaluation diverges in part from the above method since the main focus has been on evaluating Alloy from the start of the work. Thus the step of identifying a problem and proposing a solution is heavily influenced by the fact that a problem compatible with an evaluation of Alloy has to be found. The execution plan for this study is briefly covered in Section 5.2.

Chapters 7,8 and 9 document the main steps of the actions research. Chapter 7 is describes the practical "Do" step, chapter 8 focuses on the evaluation and 9 on learning.

During the evaluation, daily records were made to document the process. These records note what was done each day and what conclusions were reached in the process.

5.2 Evaluation Plan – Practical

The practical goal of this work is to evaluate whether Alloy is useful for improving a software design, and thus in extension, improving the process of software development. The problem identified in the process of software development at Teleca in general and the project Alloy was evaluated in specifically (see Chapter 6 for details about the project) is that design and structure may suffer due to tight time constraints, demanding short development cycles. This forces development teams to rush to implementation, and can lead to problems with software quality. The problems in quality might not be serious for a final user of the first version of an application, but can be more troublesome when extensions and modifications are to be made upon code that has been "rushed".

Alloy is used to study the design of software that is part of the above mentioned implementation process. A piece of software part of a larger system is studied with Alloy. The software is tested and nominally working, but may need to be refactored[2] in order to improve its overall quality. The question is: Can Alloy help the process of detecting problems and suggest improvements in the design? This is evaluated by implementing an Alloy model of the software, and the success is measured by evaluating the data items defined in Section 5.3.

[2] "To refactor" is a verb used in software development to mean redesign and rewrite of an existing piece of software using the old version as a starting point. The final version after a refactoring may but need not contain old code.

5.3 Evaluation Items

Three data items vital to the evaluation are identified:

- **Cost** deals with the observed cost of utilizing Alloy. This item will mainly be measured as quantitative data and answer questions like: "How much time was needed to achieve X?"
- **Practical results** deals with the observed benefits, and is mostly qualitative. It will answer questions such as: "What improvements to the implementation were identified thanks to Alloy, and (how/why) are they significant?"
- **Generality and validity** describes the generality of the field of study. That is, what properties are specific to the studied design, and how valid can the results of this study be for similar application?

6. PROJECT BACKGROUND

The software studied in this project is under development at Teleca Mobile's Malmö office in Sweden. Teleca Mobile supplies software solutions and services to the mobile device industry worldwide. Teleca Mobile develops software solutions for mobile phones, and has developed versions of many common mobile applications in their software framework Obigo.

The software project studied is an implementation of *PoC* (Push-to-talk over Cellular) that is part of the larger Teleca IMS (IP Multimedia Subsystem) project.

PoC is a service that allows a user of a cell-phone to communicate with one or more other cell-phone users by pushing a button and talking. However, at most one person in a PoC-session may talk at one time. Other users wishing to speak will have to wait for the active speaker to finish.

There are a number of problems to solve when building a PoC solution. One problem is setting up and using a connection for data transfer, and another is to encode and decode voice data to a format that can be effectively sent over a mobile data connection. In the design studied in this project, these two problems are closely linked together. The reason for this is to allow encoded voice data to be sent by the data connection with little system overhead, and vice versa, received data to be quickly decoded.

Alloy is used to study the dynamics of controlling the interaction between sending and receiving data on one side, and decoding and encoding sound for transport on the other. A simple schematic of the design can be seen below in 6.

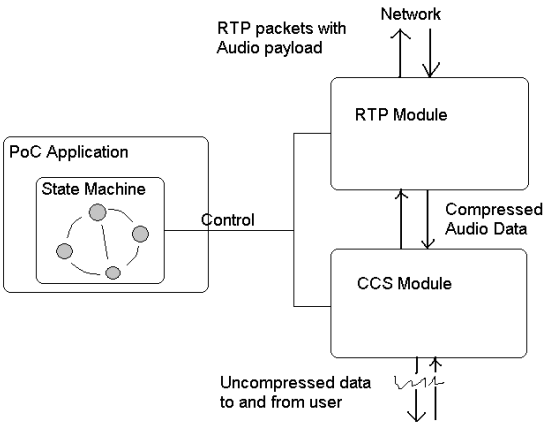


Figure 1.The studied state machine in its application context

The design studied in this evaluation is the solution to a control problem. A state-machine has been implemented in C-code. The state machine is used to communicate with and control two concurrently working software modules that are used by the PoC service. The communication is handled with signals, usually in the form of a request-response pair, where the state machine sends a request to either of the two modules and then receives a response indicating if the target module has attained the requested state. The state machine also receives driving control signals from within the PoC module. The part that Alloy is applied to is the functionality of the state-machine controlling the behavior of the two support modules.

The RTP module manages real-time communication of data and is an implementation of the *RTP* (Real-time Transport Protocol). The CCS module handles codecs[3] for sound compression and decompression and works with streams of data to and from an RTP connection and a microphone. These two modules work closely together to provide an efficient data-transfer service, and each depend directly on the other being in the correct state to function.

The "state" of the state-machine holds no real data in itself. Instead it can be seen as a record of communication between modules and within the PoC module. In this record the state is advanced depending on the content of incoming messages. An incoming message can be a response from one of the two external modules, which tells that that module succeeded or failed in attaining the requested state. It can also be a control message from inside the PoC module.

7. ALLOY MODEL IMPLEMENTATION

The Alloy models of the system were not conceived and implemented in their final form from the start of the study.

[3] A codec is a way to encode a media file, for example an image or an audio or video stream so that it takes up less raw space and can be stored or transferred more easily, but still be accessed again in its original form by decoding. Some examples: JPEG, picture encoding, MP3, audio encoding, XVid, video stream encoding.

Instead, they developed gradually. Each new model was based on the experience gained from implementing the previous form. To give a demonstration of this process, the system model is represented as several distinct entities representing the major stages in this evolution. The models are ordered by the time they were constructed, with the earliest "draft" models first.

7.1 Model input

Three distinct sources were used as input when creating the models:

- Design documents
- Developer input
- Code

Each source had its influence on the created models, but they varied in importance. The design documents gave an overview of the structure of the studied state machine and the process interactions, but lacked internal detail. This meant that the design documents were useful as a starting point, but insufficient to base the later models on. The code provided a lot of details about internal properties and a senior developer responsible for the implementation was available on site to provide much complementing information.

7.2 Model 1 – Basic State Machine

An important aspect to note about modeling in Alloy is that there is no explicit mechanic to describe a state machine. The first distinct model created was therefore essentially an experiment with the aim of finding a representation of a state machine that would be well suited to model the PoC design. To reduce model size, the choice was made to build a state-machine representing the state of only the codec module for the initialization part of the design.

Two abstractions were immediately needed; a way to represent a specific state, and a way to represent a possible state transition. For the first, the Alloy concept of the signature was used to represent the distinct states. All the signature atoms representing the state were part of a set of signatures called simply "StateT". This made it easy to represent transitions, as each state signature could contain a binary relation from itself to a set of states that represented the possible transitions, like this:

```
abstract sig StateT
    // A super set for all states
{
    links : some StateT
    // All states have at least one link
}
```

An actual state would then be represented as an extending signature like this:

```
sig CODEC_STATE_INIT extends StateT
{
    links = CODEC_STATE_i + CODEC_STATE_j +
…             + CODEC_STATE_n
    /* Where i,j…n would be the names of
       individual states and this could
             include INIT itself */
}
```

With the above information available for all states, it was also very easy to describe the change of state over time by the fact:

```
fact Transitions {
    all s: State - ord/last() | let s' =
                      ord/next (s) |
       s'.currentState in
             s.currentState.links
}
```

As is quite obvious however, this model is virtually useless for any kind of analysis, because the model can only react in a predictable and strictly flawless fashion. It reacts flawlessly in the sense that all reachable states must be in the set of good states, because it makes no sense to add links to extra erroneous states. If we knew about any erroneous states we could obviously remove them without resorting to use of Alloy!

However, the model has one redeeming quality, and that is that can easily produce a good overview of the structure of the state machine. See 7.2 below.

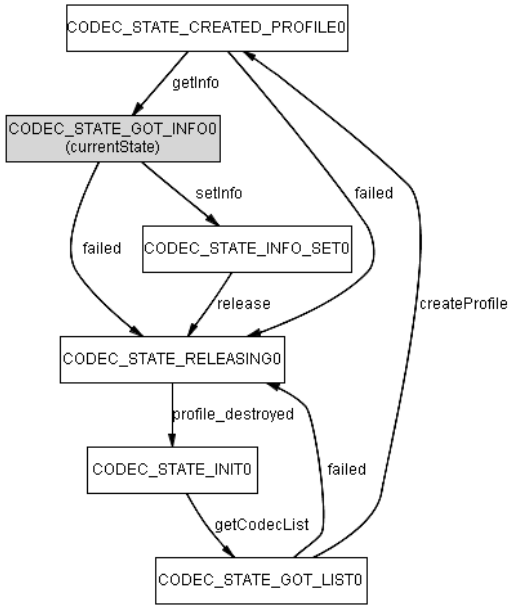


Figure 2.A state machine representing the state of the codec module

7.3 Model 2 – Signal Driven State Changes

While Model 1 above can produce a good graphical representation of states and transitions in the model, its actual functionality is extremely limited. To closer represent the actual system, the concept of signals was introduced in the second distinct model. Signals represent the input to and output from the system. Also, if the system is in state S1, an incoming signal Sig1 should prompt one response, while if the system is in state S2, another response or even an error may occur.

A new set of relations were added to the signature for each state to describe what signals the state can handle:

```
sig CODEC_STATE_GOT_LIST extends StateT {}
{
    links = CODEC_STATE_DESTROY_SESSION   +
            CODEC_STATE_CREATED_PROFILE
    signals = Sig_getCodecList ->
    (Sig_createCodecProfile + Sig_terminate)
}
```

The new field `signals` is a binary relation between two sets of signals. The first set in the relation corresponds to the space of incoming signals that this state can accept, and the second set corresponds to signals that may be sent by the system in this state. In conjunction with the old field `links` this is enough to fully determine how the system will respond to any signal in any state.

To facilitate the transitions in this model, a slightly more complex set of rules are used than in Model 1. First of all, a basic fact separates the different states in order, much like before. But now, for each state where a codec signal is involved, it makes sure that a predicate is true, rather than simply check that the state in s' is reachable from s:

```
fact StateTransitions {
    all s: State - ord/last[] | let s' =
                   ord/next [s] |
    s.outgoingSignal in CodecSignal =>
       CodecTransitions[s,s']
}

pred CodecTransitions (s,s': State) {
    s'.currentState in s.currentState.links
    s'.incomingSignal = s.outgoingSignal
    s'.outgoingSignal  in  s'.currentState.
       signals[s'.incomingSignal]
}
```

In the predicate `CodecTransitions` three items are checked that describe the transition between two codec states. First the old check that s' is reachable from s is performed. Then the incoming signal (defined as a response) to s' is set to the same as the outgoing signal (defined as a request) from s. Then the new outgoing signal is chosen from the possibilities available in the set of signal relations in s'.

This means that the model is good in the sense that forgetting to include a state transition or signal in the definition of a state is easy to spot with some reachability tests in Alloy like the following:

```
pred showStartToInfoSet () {
    let s = ord/first [] |
        s.currentState = CODEC_STATE_INIT
              some s':State |
        s'.currentState=
CODEC_STATE_INFO_SET
}
run showStartToInfoSet for 6
```

If the system is meant to be able to go from the first state INIT to the state INFO_SET in 6 transitions or less, and Alloy cannot find an example when it tries to analyze the above, there is something wrong with model, design or both.

Unfortunately this model suffers much the same weaknesses as Model 1. While Model 2 is more accurate because it includes signals, it is also completely static, and an incorrect state or state/signal pair would have to be added explicitly for the system to show an error.

7.4 Model 3 – Resource Model

The third model, a *resource model*, keeps the concept of signal driven state changes, but does away with most of the other mechanics from the past two models. In the resource model, a state is no longer a signature in the Alloy model, but rather a collection of resources. 7.4 below depicts the conceptual change in representation of state.

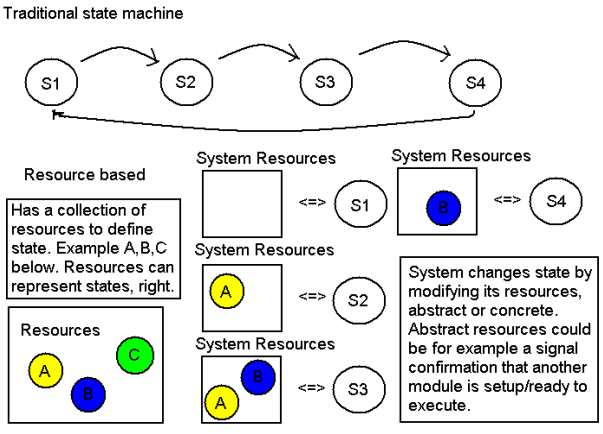


Figure 3.The resource model

In Alloy state is now represented with the following code:

```
sig State {
```

```
    codecState : one CodecResourceState,
    incomingSignal : one Signal
}
```

Where `CodecResourceState` is simply a set of resources:

```
sig CodecResourceState {
    myResources : set CodecResource
}
```

A transition between states can thus be described by additions or deletions of resources from the current state. In the code for this model, a predicate for determining if the machine is in the state "info_get" looks as follows:

```
pred codec_state_info_get
(cS:CodecResourceState) {
    cS.myResources = CodecListResource +
        CodecProfileResource +
              CodecInfoGetResource
}
```

This means that the set `myResources` must contain exactly the three sets of resources above and nothing else, for the machine to be in the state "info_get". As with the state and signal signatures before, the resource sets have a multiplicity of one to allow the use of the label for a set and its sole member interchangeably. For example, `CodecListResource` is a set, but it consists of only one atom, so saying that `CodecListResource` is a part of another set is the same as saying that the one atom that makes up the `CodecListResource` is a part of the other set.

With this new way of representing state, dealing with incoming signals becomes more complex. Now there is no set representing the links from one state to others, instead we have to check what state the machine is in, and then try to see how an incoming signal affects that state. To solve this problem, the resource model contains predicates that can be seen as signal handlers. They each represent the ability of the machine to react to one type of signal. To choose what handler to use, a simple mapping from signal to handler predicate is introduced:

```
pred CodecTransitions (s,s': State) {
    s.incomingSignal in SIG_getCodecList =>
        incomingSIG_getCodecList[s,s']
    s.incomingSignal in
        SIG_createCodecProfile =>
        incomingSIG_createCodecProfile[s,s']
    /* ... For each possible signal ... */
}
```

For this model, as before, `s` represents the current state, and `s'` the next state. There is one signal handling predicate for each possible input signal and each type of signal is represented by a set that is disjoint from other types of signals. A handler for the signal `SIG_setCodecInfo` is implemented as follows:

```
pred incomingSIG_setCodecInfo(s,s':State) {
    let cS = s.codecState | {
        codec_state_info_get[cS] =>
        {
        s'.codecState.myResources =
    cS.myResources + CodecInfoSetResource
        } else {
        s'.codecState.myResources =
        S.myResources
        s'.incomingSignal =
           SIG_destroyCodecProfile
        }
    }
}
```

Here we can see that the signal handler takes two states, the current, `s`, and the following, `s'`. It then looks at the `codecState` of the current state, which represents the resources in the resource model of the state-machine, and checks if we are in a legal state to accept this signal. There are some dependencies in the real system that must be met for the signal to be legal. One such dependency is that `getInfo` must come before `setInfo`. This means that for the signal above to be legal, we have to be in the state were we have received the resource `getInfo`.

A predicate is used to determine if the machine is in a specific state. If we are in the correct state the setInfo resource is added to the codecState for the coming state **`s'`**. If we are not in the correct state however, we chose to terminate by forcing a termination signal as input to the next state.

7.5 Model 4 – Full System

The fourth model takes the step into full functionality as envisaged by this project. It turns out that the resource model described in subsection 7.4 is quite easy to work with in Alloy. So while the code swells to over 400 lines in this model, and over 500 with various tests included, the complexity of the mechanics does not increase by much. The main difference compared to the last model is that it can handle all the signals that affect the state of the model. This means that it includes the state of not just the codec module, but also the RTP module and the internal state for the PoC module.

One minor abstraction added to that of Model 3, is that resources are divided into subgroups according to the different modules whose state they represent. This does not change the functionally significantly, as sets of resources are still simple sets of named atoms, but it does allow some predicates to be subdivided into less complex constructs in a natural way. It also allows an easier overview of results graphically, for much the same reasons.

The subdivision is represented in Alloy by the three resource variables that make up most of the new ordered **State** variable:

```
sig State {
    codecState : one CodecResourceState,
    rtpState : one RTPResourceState,
    ptsState : one PTSResourceState,
    incomingSignal : one Signal,
    shuttingDown: lone Marker
}
```

Each of the **ResourceState** variables represents a set of resources, and together with the incoming signal, fully describes the functional state of the system. The one exception to this is the system during shutdown. During shutdown, the system can be in several states that are normally illegal. This is ok in the real system, because once shutdown is initiated, it is forced to complete before new attempts to use the system are made.

While the predicates that control state transitions are somewhat more complex, they differ little from those presented in the resource model above in subsection 7.4.

This model contains some abstractions that are testable in Alloy. The tests developed concern mainly reachability and soundness properties. In this case, reachability refers to the property that from a starting state it should be possible to reach a specified end state and back again. Soundness refers to the property that regardless of starting state it should not be possible to reach an unsound state from a sound state.

Building a reachability test turns out to be simple:

```
pred showLongTransit() {
    let s = ord/first[] |
        all_state_init_setup[s]
    some s:State | rts_destination_added[s]
}
run showLongTransit for ...Scope...
```

Here we specify that the first state in the ordering will be in the initial state, as defined by the predicate **all_state_init_setup**. Then we say that some **State** in the space of all the generated states should match the predicate **rts_destination_added**. That is, we should reach the corresponding state during the simulation. When the simulation is run, if an example is found, then it is proven possible to reach the requested state from the first.

In this case, the requested state corresponds to the last state, when all resources are ready. This will take quite a few steps, but we can use the Alloy visualizer to see exactly what steps the simulation took to reach its goal.

There are a few more tests regarding reachability included in the model, but the principle is always the same as for the above test. The test of soundness requires a somewhat different technique. Now it is not enough to know that it's possible that the system is sound, it must be possible to be certain that it cannot become unsound during the modeled state transitions. To test this, the Alloy concept of assertion is used as follows:

```
assert all_good_state_leads_to_good_state {
    all s:State-ord/last[] | let s' =
        ord/next[s] |
    {
        stateOk[s] => stateOk[s']
    }
}

check    all_good_state_leads_to_good_state
for ... Scope ...
```

Since an assertion is used to check that all possible instances up to the scope comply with its definition, the above means that if any state **s** fulfills the requirements of the predicate **stateOk**, then so must **s'**, which is the state after **s**. Or simpler, if state *t* is a correct state, state *t+1* must also be correct. Running the code in Alloy with a scope of 2 states shows that a good state does indeed lead to another good state. A scope of 2 states is enough in this case because the starting state is not restricted, which means that Alloy will test all good states for the first state **s**. Since Alloy will also check all possible transitions for all good initial states, this should cover every single possible state transition from a good state to any other.

8. EVALUATION RESULTS – COSTS AND BENEFITS

This chapter describes the results of the study from a cost versus benefits perspective. Section 8.1 covers Cost and Section 8.2 Practical results, as the evaluation items identified in Section 5.3.

8.1 Cost of Deployment

The initial time set aside for and spent on learning about and practicing with Alloy was about 4.5 weeks in total over the various phases of the thesis work. This time includes reading Jackson's book about Alloy [6] and working on its exercises, but does *not* include time spent using it for work directly related to the evaluation and thus implementation of the models in Chapter 7.

During the implementation phase about 4 weeks was spent working with Alloy. Roughly half of this time was spent not on writing or debugging the Alloy models in chapter 7, but rather figuring out how to do specific things in Alloy directly related to the design. That is, how to solve a problem might have been clear logically, but exactly how to do it in Alloy was not clear. The time spent to resolve these issues is also considered time spent learning Alloy.

The time spent learning about the evaluation project and its design was about 2 weeks in total. Together with the above, the following table could be said to describe the total cost in time to achieve the results in chapter 7:

Table 1 – Time spent learning Alloy and the project and time spent on implementation

	Alloy	Project	Implementation	Total
Time (weeks)	6.5	2	2	10.5

The numbers above in Table 1 represent respectively time spent learning Alloy, gaining understanding of the project and model implementation. While over ten weeks spent for the limited results achieved in this thesis can be considered a huge price to pay, for an experienced Alloy user already working in a project, and thus intimately familiar with it, this time could probably be reduced to just the 2 weeks needed for implementation. This can be corroborated with the notes from the implementation diary (briefly described in Section 5.1), where implementation is often described as swift, while frequent and long stops occurred when how to do something had to be figured out. However, it is not clear how much the process of figuring out how to model something improves with experience.

It can take quite a lot of time to learn Alloy and acquire enough skill to be effective. Significant time can also be needed to understand an implementation good enough to be able to model it (this time is of course dependent on the specific implementation and its documentation). However, actually working with Alloy and modifying models can be relatively swift and straightforward once the required knowledge has been acquired.

8.2 Practical results

The results evaluated from the perspective of suitability for finding defects, improving design and understanding design.

8.2.1 Discovering and Identifying Defects with Alloy

The practical results of the Alloy implementation were limited in this respect, and no defects were found through Alloy analysis. The first fact that no defects were found, can be due to any or all of these possible reasons:

1. The studied state machine is defect free.
2. The model is not detailed enough or it has the wrong detail to catch bugs.
3. The wrong tests are made with the model.
4. Alloy is not suited for the type and scope of problem studied.

#4 is perhaps the most relevant of the four reasons. While there is no obvious problem modeling a state machine in Alloy, a state machine it is a very simple and well known abstraction; receive a signal, move to a different state, or stay in the same state. There is simply not much to test about the pure mechanics of a single state machine. Perhaps looking at a larger system including several state machines and their dependencies would have been more relevant in this case, but then it is likely that a model checker with an explicit notation for inter-dependent state machines and optimized for that kind of complexity would have been a better choice.

8.2.2 Design Improvements and Simplification

The results in terms of improvements or suggested redesign are more interesting to look at. While constructing the models in Chapter 7 it was soon clear that having explicit states that might or might not treat incoming signals differently was cumbersome and inefficient in Alloy. Instead, it was more natural to model state as a set of state variables, or resources, which in turn made it easy to handle each signal in one place in the implementation. This lead to the suggestion of replacing the explicit state machine with functions handling each state with a more implicit state machine based on functions handling signals instead of states.

While explicit signal-handling was not implemented in practice in code, it was discussed with the person who had implemented and designed the code at Teleca, and he said that he had considered a signal handling approach himself for the original design, but choose the more basic state machine design to make sure that he could complete the implementation in time. Unfortunately, at this time of the evaluation, Teleca underwent a large reorganization, and the new design was never tested in practice.

It is possible that an Alloy model can improve confidence in the applicability of an alternate approach as above. If so, Alloy might make it possible to choose to tailor a design to a specific problem from the outset, rather using a standard model and try to adapt it later, even when high confidence of meeting a strict deadline is required.

8.2.3 Understanding a Design and Modeling it

One of the useful properties of Alloy is its ability to simulate an event or a chain of events with the help of its analysis tool. This feature is useful both when attempting to understand and to correctly model an existing design.

In this work it was very helpful in making sure that the state machine had been realized correctly, because it was easy to test both random and specific traversals of the state space. Several problems in logical expressions in the model were easily found and corrected thanks to the ability to simulate and step through its execution.

Attempting to create a model also helps to make sure that full understanding of how the system works is reached, because there is little room for ambiguities in an Alloy model. If you do not understand a design well enough to correctly and logically describe how it works, it will either not work at all, or exhibit flaws that Alloy can help find.

9. PROBLEM GENERALITY AND VALIDITY OF RESULTS

The Alloy model created in this study, for all its code and state-space size, consists of very simple abstractions. Since Alloy was meant to test software abstractions [6] and not the structure of code itself, it is perhaps not surprising that it has failed to attain greater results than those discussed in Section 8.2. A major reason for this is that a state-machine, which was the core

abstraction of the design studied, is a very simple abstraction with few testable properties. With the part of the PoC application studied in this thesis, there were undoubtedly problems to be found, but Alloy was not contributing greatly to the process. It is possible that it would be easier to produce significant results with Alloy investigating a design with more complex abstractions.

In summary, Alloy cannot be expected to be a "magic pill" to sort out basic difficulties in a design. And if a design is very basic it is probably better to skip the overhead incurred by using Alloy and work with basic problems by other more basic means (code review, code-based refactoring etc.). Alternatively, Alloy could be used to develop new design abstractions if the user has the freedom and resources to make more significant changes.

10. CONCLUSIONS

This paper describes an evaluation of an attempt to use Alloy as a generic design improvement tool for the development process at a telecommunications company. The objective of the evaluation was to build an Alloy model of a control system for the handling of communication and encoding in a Push-to-talk application. There was no specific problem formulation beyond that problems had been experienced with the design in the evaluation. Thus the intent was to study the system with the help of Alloy and see if any weakness could be found or useful improvements suggested.

What was found was that it can take a lot of time and effort to learn to work with Alloy. Once modeling was underway however, Alloy proved swift and agile to work with. But while it was possible to model a part of an existing design, only limited improvements and no bugs were found with the use of Alloy.

This need not necessarily mean that Alloy cannot live up to its potential in terms of being able to improve and verify an existing design, nor that it will fail to produce pertinent results in all contexts that have many similarities to this project. However, it seems that there are some demands on a project for Alloy to be worth considering for use in verification or enhancement. The main issue identified is that an existing design should be quite explicit and coherent for Alloy to be useful in studying it. That is, there must be some palpable and interesting abstractions in place because Alloy does not appear very useful to test the implementation of very simple abstractions such as a single simple state machine.

For future work, it is recommended that for similar evaluations, great care is taken in the preliminary phases of the work, especially if existing software is to be studied. Two things should be possible to envision before deciding on a project or piece of software to model and verify with Alloy: That it is possible to identify some clear design abstractions that can be modeled and that it is possible to identify some specific issues that can be tested about the identified abstractions. If this is not possible, another option is to have freedom and time enough to do major refactoring work on a project and use Alloy to explore new design choices. If neither of these conditions can be established however, there is a risk that the study can reach no further conclusions about Alloy than this one.

11. ACKNOWLEDGMENTS

We would like to thank Daniel Jackson and Felix Chang for helpful input on Alloy and Christopher Wong from Teleca for much assistance with the PoC implementation.

12. REFERENCES

[1] Tina Nolte, 2002, *Exploring Filesystem Synchronization with Lightweight Modeling and Analysis*, MIT Masters thesis,

[2] Edmund M. Clarke, Bernd-Holger Schlingloff, 2001, *Model Checking*, Handbook of automated reasoning, pp 1635-1790, Elsevier Publishing

[3] Ron Jeffries, Ann Anderson, Chet Hendrickson, 2001, *Extreme Programming Installed*, Addion-Wesley

[4] Holzmann, Gerard J, 1997, *The model checker SPIN*, IEEE Transactions on Software Engineering 23(5): 279-95, IEEE

[5] Havelund, Skou, Larsen, Lund, 1997, *Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL*

[6] Daniel Jackson, 2006 *Software Abstractions*, Cambridge, Massachusetts: The MIT Press

[7] http://alloy.mit.edu, front page quoted on 2007 01 19

[8] Brant Hashii, 2004, *Lessons learned using alloy to formally specify MLS-PCA trusted security architecture*, Proceedings of the 2004 ACM workshop on Formal methods in security engineering, pp 86-95, ACM Press, NY, USA

[9] Martin Höst, Björn Regnell, Per Runeson, 2006, *Att utföra sitt examensarbete*, Studentlitteratur

[10] Chris Wallace, 2003, *Using Alloy in process modeling*, Information and Software Technology 45 pp 1031-1043, Elsevier

[11] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu, 1997, *Symbolic Model Checking without BDDs*, Lecture Notes in Computer Science, Vol 1579, Berlin, Springer-Verlag

[12] Magnus Lindahl, Paul Pettersson, Wang Yi, 2001, *Formal design and analysis of a gear controller*, STTT - International Journal on Software Tools for Technology Transfer, Springer-Verlag

A simple quantitative failure prediction model

Hanna Scott
School of Engineering
Blekinge Institute of Technology
SE-372 25 Ronneby, Sweden
+46 (0)457 385800
hanna.scott@bth.se

ABSTRACT
Failure prediction has been dominated for a while by Reliability Growth Models (SRGM). SRGM's are used very late in a software project when it can only be used to govern the late testing of the software and estimate the reliability of it. The number of failures possible to provoke in software governs the time spent testing, thus it would be desirable to have a prediction earlier in a software project, to get a sense of the resources needed for testing. There are also some basic issues connected to SRGM's, like prerequisite statistical knowledge in order to understand how it works. The lack of understanding of the mechanisms and factors used to make predictions with SRGM can make its results harder to trust. In this paper a model, called the Q-I model, is presented as a complement to software reliability modeling. The model works as a first step toward introducing quantitative measuring at companies who have expert based prediction. The model a non-complex, easy to understand and easy to use failure prediction method. The paper also contains a first case study of the model in action, which shows it can predict results as close to the actual number of failures as 24%.

Keywords
Failure prediction, case based reasoning

1. INTRODUCTION
Failure prediction today is usually done by using reliability growth modeling, i.e. the failures themselves are not predicted so much as the rate of decrease among failures. The reliability growth model creates predictions during testing, which means that the use of the predictions is limited to the test phase. Some companies use fault prediction to predict the number of failures. The problem with making failure predictions using faults is that very little is known about the fault-failure correlation.

Another problem with reliability predictions is that they cannot be used early in a project, because there is no failure data to build on. The test budget is set at the beginning of a project, and the test budget is directly dependent on how many failures are found in testing. In order to set a test budget, there is a need for failure predictions at the beginning of a project.

In this paper we present a model for predicting the number of failures expected to be found in the software during system test. The model is called Q-I, and it allows the first predictions of failures to be done before there even is code. The model is of a quantitative nature, and is built on how the number of failures relate to the time spent introducing and uncovering the faults causing them. The Q-I model aims at helping people predict the future by remembering the past but without loosing focus on the current project's specific type and circumstances.

The model is a first attempt at creating as objective project size ratios as possible of the current project, based on previous projects using project similarity, and the first rough estimates made in planning. The model is to be used as decision support for people who estimate in the planning phase, and can be used again and again throughout the project as follow up.

The quantitative failure estimation model is based on percentage divisions between the efforts spent in previous project on the phases of: Planning, Design, Implementation and test. Together with the number of failures found in old projects, these percentage divisions create a structure for estimating either project phase efforts, or the number of failures that can be found in system test. Before presenting the model, some of the research in effort, fault and failure estimation is presented in the related works section. After that the model is presented, followed by an industrial case study of its use, where a customization method for the model is exemplified. The paper is concluded by a short discussion of the case study results and some conclusions that can be drawn from it.

2. RELATED WORK
This section contains research contributions which relates to methods for estimation of failures and effort. The largest two parts of this section covers the "Case Based Reasoning" techniques and different ways to perform failure estimations. Case Based Reasoning is a method used to establish similarities between cases, and failure estimation models can work as a complement to the quantitative failure estimation model presented in this paper.

2.1 Effort estimation methods
The quantitative model presented in this paper produces two kinds of output: failure estimates and effort estimates. There has been a lot of research done in the field of effort estimation, and the methods researched can be divided into algorithmic approaches, such as COCOMO, and non-algorithmic approaches, such as Background Propagation or Case Based Reasoning (CBR) [1][2]. Algorithmic approaches encompass such methods as linear regression, stepwise regression etc., and the non-algorithmic approaches are mainly divided into machine learning methods, and analogy comparison methods. There is no clear line dividing machine learning from analogy comparison since analogy comparison can be used with or without tools, where the tools

could be of CBR type, and since analogy comparison itself is sort of the generic parent method of case based reasoning type [2]. The analogy comparison approach is a generic method for comparing the characteristics of an item, against the characteristics of items in a set, to find as close a match as possible. First, CBR is a method primarily used in machine learning, but the method itself is a specific way to use analogy comparison. Second, CBR is a method of comparing new problems with old problems, in order to reuse the solutions to old problems, especially where the factors influencing the solution are obscured or in other ways unknown [3]. Finally, CBR is in other words simply an analogy comparison method used for problem solving. The analogy approach is often used in software engineering to find similarities between projects for the purpose of estimation, where the approach can be divided into three steps [2]:

1. Defining what characteristics can be used to compare projects.
2. Defining what constitutes as similarity between two projects, and with what confidence it can be called a similarity.
3. Defining how to use the known values of the old project found to be similar, to help the estimations for the new project.

Analogy comparisons between projects for the purpose of estimation are generally very computationally heavy since the datasets used are large [4], and therefore the analogy comparison can be greatly eased by the use of an analogy comparison tool like ANGEL [2] or ESTOR [4]. The performance of analogy comparisons using the ANGEL tool, measured in the Mean Magnitude of Error (MMRE) unit, ranges from 37 to 78 percent so it performs better, or at least with less varying results, than some algorithmic approaches like stepwise regression that ranges from 45 to 252 percent (MMRE) on the same datasets [4].

The Q-I model does not use an analogy comparison tool because such tools require a large case base which is one of the problems with large quantitative models that the Q-I model is trying to solve; allowing companies which lack large case bases to still be able to use their historical project measurements to help in failure estimations. The quantitative model does however use the analogy comparison technique, but without the tool support.

2.2 Failure prediction methods

The quantitative failure estimation model presented in this paper does not calculate an estimation of total number of failures for a software system, but the amount of failures testers can expect to find in a system, using stable processes. The discrepancy between all possible failures for a software system and the found failures is not addressed in this paper. The reason for including a section of failure prediction methods is to see what factors quantitative failure estimation models usually base their predictions on.

The first and most commonly known way of estimating the amount of failures is to use software reliability growth models (SRGMs), and see where the cumulative failure curve seems to flatten out. SRGMs use failure data, mainly time of occurrence (i.e. time between failures) and amount of failures (i.e. amount of failures up to time *x*), together with statistical methods for failure distributions [5]. Two of the predecessors and most basic, commonly known models for software reliability growth, is the Jelinski-Moranda model, which is based on a simple binomial distribution, and the Goel-Okumoto model, which is based on a Non-Homogeneous Poisson Process distribution. The number of SRGMs has grown a lot since the 70's, and now encompasses over 50 different models, and SRGMs can now be seen in statistical tools like Weibull++, RGA software, Relex reliability prediction, SMERFS, CASRE and so forth [6-10], making it easy to draw the conclusion that using failure data to predict number of failures that will occur in system execution has had some success.

A more recent method of predicting software failures is to use what is called a Markov Bayesian network model [11]. A widely used way to depict a system is to use graph models where the system states are depicted as nodes, and the changes in states are depicted by edges. The Markov Bayesian network model actually consists of two combined models that use the edges and graph depiction of system states and state changes, to model the probabilities of each system state in order to get a holistic view of the probability of failures occurring system wide [12]. The two models combined in the Markov Bayesian network model are called Markov chains and Bayesian networks. The special trait of the Markov chain method is that it does not take the probabilities of any other states than the current one into consideration, and the Bayesian network method is characterized by considering system state dependencies [12].

3. METHOD

In this section, the Q-I model's design is presented. The model is a loose structure of ratios between project phases, where all measurements use the person hour measurement. The model was based on the mathematical rule of division and reformulation of division. In the formula below, you can see a division and the inverse of the same division.

$$\frac{\text{No. of failures}}{\text{test hours}} = \text{Failures per hours of test}$$

$$\text{Failures per hours of test} * \text{test hours} = \text{No. of failures}$$

3.1 Method overview

The basic idea is to get ratios from an old, similar project and by multiplying them with one or more estimates of phase effort, get a rough estimate of subsequent phase efforts and the number of failures the system test phase was likely to encounter using the "company standard" test process. The use of the method requires preparation, initialization and follow-up, and was divided into six major steps:

1. The classification of old projects into project types.
2. The selection of a set of projects of matching project type.
3. The subsequent selecting of a specific historical project.

4. The gathering of available measurements from the selected historical project.
5. Initializing the model structure using measurements from the selected historical project.
6. Give the model a phase effort input, and calculate the failure estimates.

3.1.1 Step 1: Classify old projects into project types

To simplify the decision of from what historical project the constant values for the model should be gathered, the historical projects were first classified according to project type and project size. The project types were company specific, and consisted of a set of project types that represented the main development project types at the company. The division of finished projects into project types, and what those types are, is up to the quality managers at the company. The division is subjective, thus the factors to consider may vary. A lot of companies only have small case bases, i.e. sets of historical projects, so it was important that the classification should be general enough to allow similarities between projects to be found, because a too narrow classification scheme may result in no project similarities being established. For larger companies, a good example of a classification scheme for project type could be: "new development project", "subsequent release project" and "maintenance project". If the company is constantly focusing their efforts on the same type of projects in the same problem domain, it is possible that the classification is unnecessary. The important thing is that the company uses a project classification that is representative for their development.

3.1.2 Step 2: Select a set of projects of matching project type

Once the classification was made, the new project type was decided, and the entire set of old projects of the same project type was selected.

3.1.3 Step 3: Choose the historical project with the size closest to the new project

The size classification chosen is not discrete but continuous, meaning that the project closest in size is chosen, no matter how far from target it is, which is an especially good method when working with a very small case base, because it always results in, at least, one project being selected. It is well known that, as the size of a software system grows, the more failures it is likely to exhibit in execution. The number of hours spent on the project is closely related to the number of lines of code in it, because in the generic case, more lines of code means more functionality, which in turn means a longer requirements extraction, analysis and design phases. It is important to keep in mind, that complexity of the software also plays a great part in determining the size of a software project, which is what we are trying to pinpoint by letting company representatives classify both the old projects, and the project to estimate.

3.1.4 Step 4: Get available measurements from the historical project

When the project has been selected the measurements collected from it needs to be inserted into the structure of the Q-I model. Preferably the measurements should not have to be collected but should be chosen so the readily available measurements can be used. The one measurement that has to be present is number of failures found in the software during test. The rest of the measurements should ideally encompass the effort measurement of all development phases, as well as from all separate test activities, which would include: effort of planning phase, effort of design phase, effort of implementation phase, effort of unit test phase, effort of integration test phase, and effort of system test phase. It is also possible to add failure correction time in person hours as a phase even if it is spread out over the test phases.

These measurements are then used to construct ratios:

- Hours of design per hour of planning.
- Hours of implementation per hour of design.
- Hours of unit test per hour of implementation.
- Hours of integrations test per hour of unit test.
- Hours of system test per hour of integration test.
- Failures found per hour of system test.

In Figure 1 below, the model structure, the ratios are used in, can be seen.

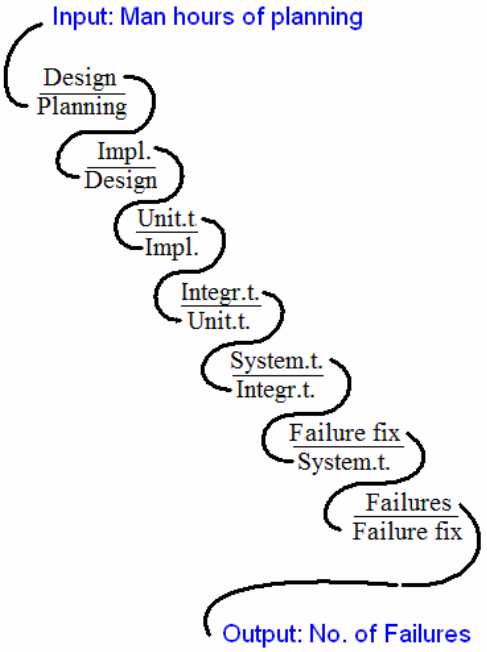


Figure 1. The Q-I model in full form.

However, the model can work with as few as two effort measurements together with the number of failures. It is important to keep in mind, that even though it is possible in reality to calculate a failure estimate based only on one effort measurement and number of failures, this option is not included in the model paths shown in Figure 1, because unfortunately the phases preceding the test activities have a weaker connection to the number of defects than the test activities. The recommendation is that at least one of the two effort measurements be of test activity

type in order not to loose too much precision in the estimate, but at the same time allowing the need of as early failure estimate as possible to determine the other effort measurement type. The measurement used for any effort should be person hours, since it is the only measurement all phases have in common.

3.1.5 Step 5: Initializing the model structure using measurements from the selected historical project

Some companies do not have all measurements seen in Figure 1 available from all their old projects. The Q-I model was built so it can be customized down to a need for only two phase effort measurements. The recommendation is that one of the phase effort measurements is a test activity effort, because it is more tightly coupled with the number of failures than most other effort measurements. Using one test effort estimate ensures some stability in the outcome of the model, in most cases.

The two effort measurements have to be complemented by the number of failures found during system test, because otherwise the model cannot produce failure predictions. A reduction down to two effort measurements will result in a less reliable prediction than if more phase effort measurements are used, and so will the use of effort estimates in phases far apart. In Figure 1 the structure of the full blown model with all phase effort measurements is shown. Figure 2 shows an example of a maximum reduced model of that in Figure 1. It is important to mention that nodes can be removed from a path—it just diminishes the reliability of the prediction. The model can also be expanded with nodes, if other effort measurements are available.

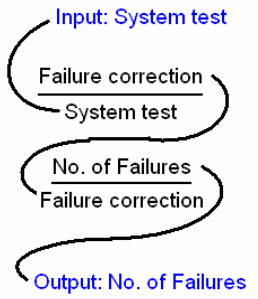


Figure 2. The Q-I model in maximum reduced form.

This means that for the maximum reduced model, the ratios "Failure correction/System test" and "No. of Failures/Failure correction" needs to be created, thus we need the three measurements from the historical project: Person hours spent on system test, person hours spent on failure correction and the number of failures reported.

3.1.6 Step 6: Give the model a phase effort input, and calculate the failure estimates

The input to the model is a project phase measured in person hours. To ease the understanding of how the model is initiated and used, we again turn to the maximum reduced form of the model. In Figure 3, all tasks are visualized.

First, the measurements from the historical project are collected, then ratios are calculated from the measurements collected from the historical project, third and last, the model is combined with the input to calculate the estimate for number of failures expected.

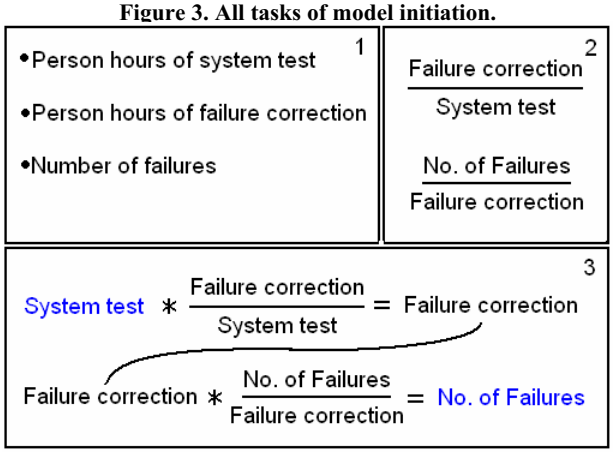
3.1.7 Step 7: Updating the failure estimates

The Q-I model can be used at several points in time during a software development project in order to accommodate new information. One example for the model seen in Figure 2 and 3 is to make a new estimate when system test is done using the actual number of work hours spent on system test. The accuracy of the input does influence the accuracy of the output, thus it is important to use what information is available.

4. Method validity

In literature, some Bayesian Belief Network research supports the existence of a correlation between test effort, problem complexity and faults detected [13]. One threat to the validity of the quantitative model presented in this paper is that the environment and its limitations forced an assumption that the project complexity can be represented by the described project classification scheme used for the case based project selection method.

Figure 3. All tasks of model initiation.



There is a risk that the characteristics (project type) are not representative when seeking similarities between projects

Effort estimates and number of failure estimates are intertwined—it is possible that even the attempt at measuring the test effort and number of faults will influence the testing process—this in turn, could make the model less accurate since the model attempts to estimate faults expected to be found using a specific test effort is based on the stability of the test process.

Data aging is always an issue when using historical data to create models and it requires that changes made to factors the quantitative model is built on—such as development process—does not impact the quantitative model too much. A model based on historical data is tied to the stability of processes from which the data was collected.

The model has no statistical validity with such a small case base. In order to have statistical validity there has to be at least eight projects to draw any statistically valid conclusions.

5. CASE STUDY

In order to make a first evaluation of the quantitative model presented in this paper, a case study of its application at a software development company was conducted. This section is divided into four sections where: the first contains some background information on the environment, the second is a description of how the implementation was done and the third contains a description of validity issues identified for the case study. The result of the case study is presented in the last section.

5.1 Background

The company participating in the case study had approximately 150 developers at the start of this study, and the company develops mobile platforms and applications. The company development processes were fairly stable at the beginning of the study, but changes to processes are becoming more common. The test process consisted of; testing of integrated features, regression test, system testing and usage based testing. The project selected for the case study evaluation of the quantitative model presented in this paper, is a subproject of a medium-sized project, and was mainly selected because of two reasons: primarily because the start of the project was close in time to when the model was ready for evaluation, and secondly because it was a project which seemed to have a representative mix of new development and old code. This was important because it meant it was an average company project, thus if it worked in that project, it was more likely to work in other projects at the company as well.

5.2 Implementation of the model

The case study was divided into seven steps starting with the classification of old projects and how an old project was chosen to initialize the quantitative model. Thereafter the model structure was modified to fit the measurements available from the historical project chosen. Then the initialization of the model was described and how the estimation calculation was conducted. The result of the case study is presented in a separate section.

5.2.1 Step 1: Classify old projects into project types

To decide from what historical project the values for the model were to be collected, the three historical projects from which data was available were classified into three different project types: Platform development, platform functionality changes, and platform application development. The classification of the projects was mainly done by company management personnel.

5.2.2 Step 2: Select a set of projects of matching project type

The new subproject chosen was found to be of type "platform application development" by the company quality manager, so therefore the data for the model structure was chosen from the only historical project of that type.

5.2.3 Step 3: Choose the project with the size closest to the new project

Because the company only had data available from three old projects of different project types, there was only one project selected when matching project type of the new project and old projects—this also meant that the step of size matching was not needed.

5.2.4 Step 4: Modification of the model structure

The measurements available at the company are not directly represented by any of the paths in Figure 2. The reason was that one of the paths needed to be shortened to create a direct connection between the measurements the company had available. The easiest way to do this was to create a path according to the method used to create the paths presented in Figure 2. The path was created in the following way:

1. The project phase effort measurements the company had available, for the company we studied were:
   a. PreFC, which was what they call everything that precedes "functional complete", so in other words all person hours spent up until the start of system testing.
   b. System test, which was the testing of the entire system according to the function specifications.
   c. Failure correction, which was not a phase at the company, but rather intertwined with system test. It was however possible to use, since the person hours were reported to a separate budget for this task.
   d. The number of failures found during system test and user tests. Only failures found to be failures the customer is likely to encounter have been included.
2. The measurements were sorted into chronological order, with the phase that came first in the project, on top. All those measurements were to be the bottom-halves of the ratios. In Figure 4 these are: PreFC in the top ratio, system test in the second to top ratio, and failure correction in the second to last ratio, and the number of failures is alone at the bottom.
3. Each of the measurements was given a roof in the form of the divisive line seen in all ratios in Figure 4.
4. Then the closest following phase effort measurements available were put on top of each of the measurement that had a roof. In Figure 4, system test closest follows the PreFC phase, meaning that it forms the top for PreFC. Failure correction closest follows system test, so it forms the top for system test. Number of failures can only be extracted correctly at point of delivery, so therefore it forms the top for the failure correction measurement.

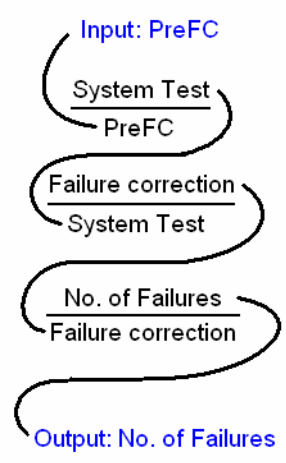


Figure 4. This case study's specific model structure.

The result of the path creation for the case study measurements can be seen in Figure 4.

5.2.5 Step 5: Insert the measurements from an old project into the model structure

The historical measurement types which were available from the historical project were:

- Hours of implementation per hour of system test.
- Hours of implementation per hour of system test failure fix.
- Hours of implementation per failure.
- Hours of system test per failure.
- Hours of system test fault fix per failure.

The measures taken from the historical project were put into the model structure. Once the nodes had been created, each node ratio was calculated by dividing the upper measurement with the lower measurement.

5.2.6 Step 6: Calculate the failure estimates

In order to produce a result from the model, an effort estimate was needed. The input effort estimate was a coarse effort estimate for the implementation phase of the project which was produced by the project management at the beginning of the planning phase, in the same way that they usually do, through subjective estimation. This of course gives the result the model produces an element of possible inaccuracy that needs to be kept in mind when making comparisons of the result to some other estimate or results.

As soon as a rough estimate of the first of the measurements was available, in this case the "preFCPreFC" effort estimate, usage of the modified path was begun. This is the formula for how the calculations were conducted:

$$\text{PreFC} * \frac{\text{System Test}}{\text{PreFC}} = \text{System Test}$$

$$\text{System Test} * \frac{\text{Failure correction}}{\text{System Test}} = \text{Failure correction}$$

$$\text{Failure correction} * \frac{\text{No. of Failures}}{\text{Failure correction}} = \text{No. of Failures}$$

5.2.7 Step 7: Updating the failure estimates

The estimate the model produced could be updated as soon as more information about the values of the measurement types became available. This meant that the estimates could be recalculated at least when the project phase changed, to give as correct a result as possible. The update was done by replacing estimates with actual values when multiplying the ratios.

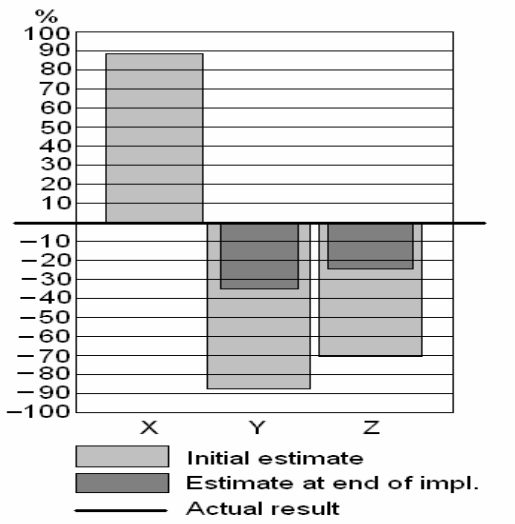
5.3 Threats to the case study validity

The largest validity issues of the case study were the classification scheme and the length of the path chosen. The classification of historical projects at the company could contribute to making generalizations harder because there were basically one representative historical project for each project type, meaning that it was impossible for external people to tell if this means that the classification was so hard that it would continue to create project type categories for each new historical project which was added to the case base or not. In turn, this raises questions on whether or not all companies would classify their projects in the same manner, thus causing the situation where a historical project match cannot be found to initialize the model structure.

The length of the path chosen, pinpoints the weakness of the model structure itself, but also raises questions of whether or not the results of the case study evaluation can be generalized further than to exactly the same modified model structure when in use at other companies.

5.4 Case study results

The result of the model is presented as relative to the deviation of old subjective predictions at the company. The comparison was made to subjective estimates in two old projects.



The first old project, which was called project *X*, was estimated without an estimation model, using expert subjective estimates. The initial estimate was 88.4% higher than the actual resulting number of failures. It is possible that intermediate estimates were made in follow up, but they could not be traced in the documentation. Project *Y*, which the second project was called, underestimated the initial estimate with 87,5%. The estimate for number of failures was redone six times before the end of implementation, and the estimate became more and more accurate compared to the actual number of failure each time, until the last estimate, only underestimated by 35.1%.

The estimates in the case study were done only twice—once at beginning of the planning phase, and once at the end of implementation. We will call the project used in the case study project Z. The estimate made during the planning phase, using the experts' subjective estimates of the size of the project until end of implementation, underestimated by 70.4%. The estimate at end of implementation showed that the project had been expanded. There were no data available that could tell us how much of the change in time was due to the expansion decision, and how much, if any, was due to over- or underestimation by the experts in the planning phase. The second estimate, made at the end of the implementation phase, now with numbers on the size expansion of the project, only underestimated by 24.0 %.

6. DISCUSSION

The theory of the quantitative model was that it should: i) be easier to discuss its results than subjective estimates, ii) be easy to adapt to project characteristics, iii) not require large amounts of historical projects, and last but not least iv) be easy to learn and easy to use. The case study has shown that it was easy to use, but perhaps not as easy to understand as hoped for. It was relatively easy to adapt to project characteristics, but choosing the characteristics to take into account was a lot harder than expected. One of the largest problems with the models has been shown to be the input. If the input estimate is inaccurate, the model's result will be inaccurate. The problem of "bad input data, bad output data" is however a well known problem, for instance, if a reliability growth model is used together with failures who are not collected from the type of testing that imitates actual use, the reliability prediction will not be correct [5].

The Q-I model is change sensitive, changes to test techniques is what the model is most sensitive to, since testing has the closest connection to the ratios containing number of failures. However, changes from the level of development process down to the level of work environment changes can affect the model, and as of yet, little is known about how different changes affect the model. In addition one might add that changes to already established test techniques is only done carefully and in most cases while running the new technique in parallel with the established one. This way one will be able to receive data and compare the two test techniques for some time and, hence, better face changes to test techniques which the model is sensitive to.

The model seems attractive to people who estimate because once the model has been customized and initialized it calculate numbers which are easy to use, which was part of the reason why the company chose to use the model instead of their normal estimation method during the evaluation project. The company has now chosen to use the model in two more projects, and the personnel who perform the estimation seem pleased with this first test run.

7. CONCLUSIONS

It is hard to draw any conclusions of the method's result validity based on just this study. The greatest achievement for now is that the Q-I model is being used, so that it can be further evaluated in the future. However, some more general conclusions can be made. The Q-I model does not require the user to have statistical knowledge in order to make predictions. The Q-I model is unusual in its construction because it conforms to the reality faced by software companies in terms of data available, both in terms of amount of cases as in types of data. In most other models for failure estimation the companies have to conform to the model rather than the other way around, so the flexibility of the Q-I model input has been well received.

8. ACKNOWLEDGMENTS

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

9. REFERENCES

[1] Lionel C. Briand and Isabella Wioczorek, *Resource Estimation in Software Engineering*, Wiley and Sons Inc, 2002.

[2] AR Gray, SG MacDonnel and MJ Shepperd, Factors systematically associated with errors in subjective estimates of software development effort: the stability of expert judgment, *Sixth International Software Metrics Symposium proceedings*, 1999, 216-227.

[3] Ross Jeffery, Aybüke Aurum, Claes Wohlin and Meliha Handzic, *Manageing software engineering knowledge*, Springer Verlag, 2003.

[4] Fiona Walkerden and Ross Jeffery, An Empirical Study of Analogy-based Software Effort Estimation, *Empirical Software Engineering*, 4, 2, 1999, 135-158.

[5] Michael Lyu, *Handbook of software reliability engineering*, McGraw-Hill, 1995.

[6] William Farr, SMERFS - Statistical Modeling and Estimation of Reliability Functions for Systems. 1982, Available from: http://www.slingcode.com/smerfs/.

[7] John Musa, CASRE Software Reliability Engineering program, 2004, Available from: http://members.aol.com/JohnDMusa/CASRE.htm.

[8] Relex Reliability Studio, 2006, Available from: http://www.relex.com/products/index.asp.

[9] Weibull ++ 7, 2006, Available from: http://weibull.reliasoft.com/.

[10] Reliasoft RGA 6. 2006, Available from: http://rga.reliasoft.com/.

[11] C.G Bai, Q.P Hu, M Xie and S.H Ng, Software failure prediction based on a Markov Bayesian network model, *Journal of Systems and Software*, 74, 3, 2005, 275-282.

[12] J.B Durand, O Gaudoin, Software reliability modelling and prediction with hidden Markov chains, *Statistical Modelling*, 5, 2005, 75-93.

[13] N Fenton and N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, *IEEE Transactions on Software Engineering*, 26, 8, 2000, 797-814.

A Method for Assessing and Improving Processes for Capacity in Telecommunication Systems

Kristian Sandahl
Linköping University
Dept. of Computer Science
SE-58183, Linköping, Sweden
+46 13 281957
krisa@ida.liu.se

Mikael Patel
Ericsson AB
R&D PM&T
SE-58112, Linköping, Sweden
+46 13 284665
mikael.patel@ericsson.com

Andreas Borg
Linköping University
Dept. of Computer Science
SE-58183, Linköping, Sweden
+46 13 282869
andbo@ida.liu.se

ABSTRACT

Capacity in a telecommunication system is highly related to operator revenue. As a vendor of such systems, Ericsson AB is continuously improving its processes for estimating, specifying, tuning and testing the capacity of delivered system. In order to systematize the process improvement activities Ericsson AB and Linköping University joined forces to create an anatomy of *Capacity Sub Processes (CSPs)* to improve. The anatomy is a result of an interview series conducted to document good practices amongst organizations active in capacity improvement. In this paper we describe five different processes in terms of how far they have reached in their process maturity according to our anatomy and show possible improvement directions. Three of the processes are currently in use at Ericsson, one is the union of good practice of the organizations in the interviews, and the fifth is the OpenUP/Basic process which we used as a reference process in earlier research. The result mainly confirms the order of CSPs in the anatomy, but we need to use our information of the maturity of products and the major life cycle in the organization in order to fully explain the role of the anatomy in planning of improvements.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *life cycle, software process models, software quality assurance.*

General Terms

Management, Measurement, Performance, Verification.

Keywords

Capacity, non-functional requirements, process improvement, anatomy.

1. INTRODUCTION

The capacity of a software product is often an important success factor and with that a critical issue in large-scale software engineering. In an Ericsson context, a true challenge is to provide systems with lowest cost per subscriber and transaction, but also with highest possible availability and at the same time allow for scalability, that is, network size and number of subscribers to grow. Delivered systems must meet the needs of both today's networks as well as tomorrow's, which means that more capacity is always needed, both in terms of bandwidth, system size, and transactions per second. Thus, improving capacity is an issue during the entire life-cycle of the system and within each development project.

However, most of the prevailing processes for large-scale software development, such as RUP [6], are primarily concentrated on delivering functionality. This means that capacity, and other non-functional requirements (NFRs), often need to be addressed by other means. The NFR Framework [3], for example, provides information on how to refine performance requirements to design solutions, and techniques on how to predict performance and make predictions meet reality are presented within the concept of Software Performance Engineering (SPE) [9].

Ericsson AB and Linköping University conducted an interview series [1] that focused on capacity in order to identify good practices and provide background knowledge for capacity improvement. It revealed that capacity requirements are well documented among other "characteristics" in 1000-pages documents. On a high level and time scale capacity requirements are well known, but challenges are:

- to formulate, analyze, and refine the requirements in terms meaningful for design, implementation and testing;
- to ensure the feasibility of the requirements given the constraints from the previous release and its test results, i.e. change impact analysis;
- to create tests that can give rapid feedback to designers on low levels; and
- to ensure that all important information is present when decisions about requirements are taken.

The analysis of the interview material and the good practices that had been identified led to the formulation of 19 *Capacity Sub-Processes* (CSPs). These were arranged in 4 different *Capacity Sub-Process Areas* (CSPAs), intended to cover all important aspects and roles when developing for capacity, as a first step towards an "Anatomy of Capacity Engineering" (ACE).

There is current work [2] on implementing the results of the interview series as a method plug-in to the Eclipse Process Framework (EPF) [5] as an extension of the OpenUP/Basic [8] software process. EPF is a Software Process Engineering Metamodel editor/code generator built on top of the Eclipse Framework. The construction of the method plug-in contributed to a deepened understanding of the relationships among CSPs, which makes it easier to evolve the CSP model into an anatomy of capacity engineering. The plug-in also acts as a pedagogical tool to show how other processes such as those reviewed in this research may be extended with the CSPs. The anatomy allows us

to both assess processes but also point to a set of vital few areas to focus on in the step of processes improvement.

In this paper we present the CSPs, the anatomy, and, by carrying out three case studies, how it can be used to assess software developing organizations' abilities to develop for capacity.

The remainder of this paper is organized as follows. Section 2 provides the background regarding context and methodology and Section 3 describes the CSPs and the anatomy. Section 4 presents the case studies and their results, which are analyzed and discussed in Section 5. Finally, conclusions and future work are summarized in Section 6.

2. BACKGROUND

2.1 Capacity and anatomy at Ericsson

Ericsson develops large-scale mobile telecommunication systems for a global market. Capacity is among the most important quality factors and can be described as the maximum number of simultaneous subscribers that use a mobile telecommunication system (compare to [4], p. 317). Maximizing throughput (transactions per second) is of higher interest than minimizing response time. Response time is an upper limit for what is acceptable for a single transaction, and maximizing capacity is then all about maximizing the number of transactions within the response time limit. Hence, capacity is a narrower concept than the often used performance.

Anatomies are popular at Ericsson and are used in different planning situations. In practice, an anatomy is often represented as a directed two-dimensional graph showing dependencies between items. A typical example is dependencies between work-items to plan what can be done in parallel and discovering critical parts. We have used the anatomy concept to help process improvers selecting the next process area to attain.

2.2 Methodology issues

The Capacity Sub-Processes (CSPs) are empirically grounded in the sense that they are the result of an analysis from an interview series on capacity [1]. The series involved 17 practitioners at 4 different Ericsson sites in Sweden, and the analysis of the interview material resulted in 19 CSPs that are described in the following section.

The anatomy is made by us and is a result of our analysis. The data from the cases C2, C3 and C4 are mostly collected by reading the process descriptions from the Ericsson intranet, complemented with some e-mail communication with the process owners. No process owners have had the opportunity to comment upon our description. The assessments may be viewed as a set of expert reviews on available processes without contact with process owners and practitioners. This has both its advantages and disadvantages. The processes may both miss actual activities in projects or not used in full detail. The next step of investigation may be to assess actual practice in the reference organizations. Case study C5 is entirely based on published material from the web. We are fully aware of that the cases C2-C4 cannot be replicated without intranet access.

3. THE CSPs AND THE ANATOMY

3.1 Capacity sub-process areas

The interview series [1] provided material from which we identified and documented what practices are necessary when developing for capacity, and we identified several methods of achieving the goals and working procedures. These methods were put into a methodological context of how to develop for capacity. The analysis showed that four *capacity sub-process areas* need to be considered when developing for capacity:

1. **Estimation and prediction.** Improving the ability to estimate and predict system capacity. Requires good practices from other sub-process areas and of the execution environment measurements (such as cost of operating system calls, memory profile/access time, etc).
2. **Specification.** Specification and refinement of capacity requirements to detailed design specification and further to implementation. Requires detailed knowledge of the system's internal structure.
3. **Measurement and tuning.** This means to regard the system from a white box perspective and to observe its internal structure in order to improve capacity by the means of tuning. For example, hardware resources can be optimized in order to improve capacity without code modification, but we also mean to gather further information and tune hotspots (profiling, recompile with compiler optimization, relink to increase locality of memory access, recode hotspots, etc).
4. **Verification.** Capacity requirements are generally known at the system level and the application level. The primary concern in order to deliver the right capacity is then to verify that the requirements have been satisfied in the system to be released (or to describe needed improvements). At this stage the system can be considered from a black box perspective to be loaded with test cases.

Each sub-process area contains several recommended good practices, i.e. activities, which are all presented by title in Table 1 and further described in the following sections. Notice that the CSP TUN8 ("Continuously planned capacity activities") has not been included in our previously reported work regarding capacity [1][2], even though it was part of the interview results.

3.2 Estimation and prediction

3.2.1 EST1: Measurement of primitives

The ability to come up with reasonable estimates that can be used for the calculation of valid predictions of capacity is based primarily on earlier experience and detailed knowledge regarding the system in scope. Measurement of system primitives is, for the sake of estimation and prediction, more important than any other of the measurements that are suggested and described in the following sections. A typical example is to include the actual (measured) message passing cost in the calculations, which means that application designers get a good idea of the cost associated with the delivered platform primitives. For embedded systems, such as telecommunication systems, this would imply collecting measurements for real-time operating system (RTOS) calls, message passing operations, file storage, etc.

3.2.2 EST2: Prediction model

There are several approaches that can serve as prediction models but all require the measurement of primitives as input. At least the following approaches need to be considered:

1. *Spread sheets* provide a basic rule of thumb calculation approach to capacity estimation and prediction. Resource measurements can be combined with measurements of primitives in order to create capacity estimates for various operations and to model, e.g., overall throughput.
2. Using *queue theory* to calculate network capacity expressed in Erlang.
3. Applying a *simulation model* (e.g., *discrete event simulation* on the individual level and *Petri nets* for overall throughput) to be able to predict the capacity as virtual time. It is also beneficial to make use of stress testing tools in order to simulate system behavior: the tool can be used to deny the system the estimated resources of a new feature to be able to predict capacity.
4. Performing *data and/or control flow analysis* to be able to predict capacity.
5. *Modeling*. UML diagrams must not be used for development activities exclusively, but are well suited for estimation too. Naturally, diagrams created primarily for prediction purposes can be used in development as well. Of course, the possibilities of the *UML Profile for Schedulability, Performance and Time* [10] should be considered.

Table 1: Capacity sub-processes (CSPs)

ID	Title
EST1	Measurement of primitives
EST2	Prediction model
EST3	UML model extensions
SPEC1	Understood refinement of capacity requirements to design specification
SPEC2	Capacity budget for sub systems and downwards
SPEC3	Augmented design model with refined capacity requirements
SPEC4	Test cases per sub system designed, implemented, and executed
TUN1	Processing load measurement
TUN2	General resource measurement
TUN3	Use profiling to identify and measure code hotspots
TUN4	Use measurements to drive configuration
TUN5	Use profiling to drive capacity improvement
TUN6	Measure quality of test cases
TUN7	Involve task force
TUN8	Continuously planned capacity activities
VER1	Capacity requirements defined, communicated, and understood
VER2	Capacity test cases and test environment defined, implemented, and executed frequently
VER3	Multiple load scenarios executed
VER4	Capacity test results part of project reporting

3.2.3 EST3: UML model extensions

The idea of extended use of UML models is primarily to be able to compute elementary consistency checks on a model. Another option would be to facilitate the prediction of hotspots that will be paid extra attention to improve capacity.

There are basically three maturity levels when modeling capacity:

1. Being able to model the capacity of the current application release.
2. Being able to model the capacity of the current application release, including platform functionality.
3. Being able to model the capacity of multiple releases in an incremental development environment.

The capacity information that is most important to represent in models is size (e.g., the number of subscribers), throughput (e.g., packets/second), response time, and frequency of use. Furthermore, the most important UML diagrams to contain this information are the Use Case, Sequence, Class, State, and Deployment diagrams.

3.3 Specification

3.3.1 SPEC1: Understood refinement of capacity requirements to design specification

It is essential that capacity requirements are refined to design specifications and further to implementation in a clearly understood way. Otherwise there is an obvious risk that the most appropriate system architecture and accompanying design alternatives are not chosen. Consider the example requirement "*The radio network shall support 32000 cells. The maximum number of neighbors per cell is 64*." The numbers stated in the requirement strongly influence design choices such as hardware or software solutions, system partitioning, threads, and even data structures that can be feasible to use to hold the information, i.e., the numbers are important design information. Thus, capacity information in a general system requirement is essential for a designer to make the right decisions and must be represented in design specifications.

3.3.2 SPEC2: Capacity budget for sub systems and downwards

If capacity requirements are refined it is generally possible to specify resource budgets regarding, e.g., CPU time, memory, disk, network, and I/O bandwidth. For example, the overall time budget can be distributed over the sub systems on all levels. Consequently, functionality (such as primitives of a generic platform) needs accompanying "capacity price tags" so that application developers can perform capacity budget refinement. However, the resource budget does not give capacity solutions by its own. The primary motivation for this CSP is to distribute the resources on the sub systems in such a way that each sub system better understand that they achieve their functionality in a shared environment.

3.3.3 SPEC3: Augmented design model with refined capacity requirements

When capacity requirements have been refined to design specifications and resource budgets have been created it is possible to annotate design models with the specified capacity requirements. Examples are, in terms of UML, to add the number of subscribers in use case diagrams, time constraints in sequence diagrams, and defined multiplicity in class diagrams. An example of the latter is "1 to `no_of_subscribers`" instead of "1 to *", where `no_of_subscribers` is directly linked to the corresponding attribute in the Use Case model (where the actual number is specified). This CSP relate closely to EST3 ("UML Model Extensions") since SPEC3 is a prerequisite for being able

to model capacity and use the models for estimation and prediction purposes.

3.3.4 SPEC4: Test cases per sub system designed, implemented, and executed

If capacity requirements are refined and specified according to the above it is also easier to create good test cases. Naturally, each sub system should be tested but it is preferable that the sub systems are carefully tested with respect to the overall system requirements. The capacity of an executable unit is what really matters to customers.

3.4 Measurement and tuning

3.4.1 TUN1: Processing load measurement

Measuring processing load on various levels (processor, board, rack, system) is essential to be able to distribute processing as close to optimal as possible. Measurements can then be used to tune the system so that processing is distributed as evenly as possible among processes, threads, and load modules, and further on to distribution among several processors. For example, the entire system might act as if heavily loaded even if all but one processor have plenty of capacity available. Thus, measuring load can lead to significant capacity improvements by changes in processing distribution.

3.4.2 TUN2: General resource measurement

The reasoning from the previous CSP can—and should—be extended to embrace other resources as well: The utilization of memory, I/O resources, cache, channels, etc. Some parameters are general real-time OS attributes whereas others are specific for the platform or even the application. Examples of the latter are how long it takes for a certain message to propagate through the system and how long traffic is delayed in queuing. Measurements like these are needed to understand how things are, draw the right conclusions, and with that be able to reconfigure the system as well as possible. Measurement data from this step provides valuable feedback to SPEC2 (”Capacity budget for sub systems and downwards”) to increase the quality of the prediction model (EST2).

3.4.3 TUN3: Use profiling to identify and measure code hotspots

Profiling is a well-known and powerful way of analyzing software systems which constitutes the next level of measurement. The key issue is to find where in the code that most of the time is spent so that bottlenecks can be avoided. A typical example is collecting statistics on function calls and execution time. However, in distributed systems, other levels of profiling, such as frequency of message passing and context switches, should be considered as well. Especially in the case of embedded design as host instrumentation is not always possible. Interestingly, Ericsson already has a lot of knowledge regarding the use of profiling techniques in distributed systems in the work by Moe [7].

3.4.4 TUN4: Use measurements to drive configuration

Measuring the utilization of processors and other resources produces useful information regarding the system’s status. Such measurements can of course be used to create trouble reports, improve prediction models, or define better requirements for the next release, but they can also be used to reconfigure the current system. The knowledge regarding, e.g., processor load should drive system configuration of external parameters such as process priorities and how memory is used. This way improvements can be achieved without editing code.

3.4.5 TUN5: Use profiling to drive capacity improvement

Profiling provides information that indicates where actual changes in code can be considered. It is obvious that code sections associated to bottlenecks are strong improvement candidates for forthcoming releases and should be traded against the expected cost of implementation. However, even though a specific block has been identified as a hotspot the task of improving the code may involve other parts, parts that, e.g., are frequently called from within the block, as well. Thus, analyzing the cause and proposing remedies require careful investigations.

3.4.6 TUN6: Measure quality of test cases

When measurements have been introduced it is also of interest to measure the quality of the capacity test cases. There are several strategies to apply, but adding code coverage facilities and statistical resemblance are probably the most important. This way the verification process can be verified, and test cases can be modified to increase the code coverage level if necessary. However, in the Ericsson perspective the traffic mixes that are used to model realistic, and unrealistic, load conditions are of special interest. If they are not well-suited the value of capacity testing is limited.

Another aspect of test case quality is under which load conditions a system is tested, since it is not always feasible (or even possible) to load a system to a realistic extent. The routines vary within Ericsson from testing parts of systems with load far below realistic levels to testing whole systems with higher load than will ever be faced in reality. Naturally, the latter should be regarded as better testing quality.

3.4.7 TUN7: Involve task force

The creation of a task force consisting of specialists from various disciplines is the standard way of dealing with urgent capacity problems (and other urgent problems as well). However, organizing a task force that analyzes the system and the ongoing project should be a rule rather than an exception. It is reasonable to believe that such cross-functional specialist teams have good chances to have an even greater positive impact when not brought together to fight fires, but to simply propose improvements.

3.4.8 TUN8: Continuously planned capacity activities

Activities related to capacity should be explicitly stated in the project plan, and examples of information to include are:

- what shall be measured and how,
- how and when measurements shall be used to tune the system,
- how the planned task force shall be formed, and
- to specify the responsibilities and the activities of the task force.

3.5 Verification

3.5.1 VER1: Capacity requirements defined, communicated, and understood

Valid system level capacity requirements are fundamental for verifying that a system fulfills its required capacity, or to assess what needs to be improved. This means that the number of simultaneous subscribers, packages per second etc. must be clearly stated in the requirements. Analysts and testers need to cooperate in documenting requirements in agreed-upon artifacts, e.g., Use Case diagrams and accompanying characteristics. This is also a prerequisite of the specification CSPA.

3.5.2 VER2: Capacity test cases and test environment defined, implemented, and executed frequently

Valid capacity requirements need to be transformed into valid test cases that are frequently executed in a well suited test environment. "Frequently" is important, since recurring measurements enables the possibility to point out capacity problems relatively early. Thus, the test environment must allow testers to provide developers with rapid feedback regarding capacity test results. However, an important remark is that this does not replace the need of good requirements and design work, since measurements alone cannot provide wanted the system with required capacity.

3.5.3 VER3: Multiple load scenarios executed

The operating conditions can vary a lot for a system. To cover expected and unexpected load condition variations, test cases need to be constructed and executed for several traffic mixes. This is the best way to verify that the overall system capacity is good enough, but there are reasons to execute separate Use Cases and scenarios in isolation as well. By doing that the parts of the system that are associated to a specific Use Case are stressed heavier, and this means that hotspots and bottlenecks are easier to identify. This CSP is the test team's concern and the main problem is to provide traffic mixes that mimic the variations of reality.

3.5.4 VER4: Capacity test results part of project reporting

Introducing capacity test reporting in standard development procedures (per shipment, delivery, build, etc.) force focus to capacity testing and enables the possibility to learn continuously from earlier testing experiences. This makes the project manager highly involved in delivering the right capacity and the CSP relates closely to several of the CSPs of the Measurement and tuning CSPA (see Section 3.4).

3.6 An anatomy of capacity engineering

There are several reasons for us to construct an anatomy of capacity engineering:

- The anatomy is a clear and pedagogical approach that is widely spread within the Ericsson organization.
- The anatomy can be used to assess an organization's abilities with respect to capacity as well as to pinpoint the most suitable improvement efforts. It is likely to believe that the assessments motivate organizations to focus on improvements, and that they aid in identifying the most important improvement actions with a feasible plan to reach the next level.
- The anatomy aids in continuous improvements in a planned manner. Acquired CSPs shall be maintained in further development and not dropped after the improvement project. Focus is kept on the abilities.

The anatomy as such was constructed by further analysis of the CSPs. The relationships between CSPs were carefully considered and special emphasis was paid to which CSPs contribute to the fulfilment of other CSPs. This resulted in the anatomy shown in Figure 1, in which the fulfilment of a CSP is dependent on contributions from the CSPs along the path to get there. Thus, the arrows in Figure 1 generally means "contributes to". The anatomy states that VER1 ("Capacity requirements defined, communicated, and understood") naturally precedes capacity improvement. However, it could of course be the case that organizations practice other CSPs even though VER1 is not fulfilled.

The anatomy itself should be regarded as nothing else than a way of visualizing the CSPs, relationships among CSPs, and assessments of CSP fulfilment. Therefore, a supporting method is needed to make it an "Anatomy of Capacity Engineering" (ACE) that is useful for planning the improvement of processes with respect to capacity. The initial two steps of the method have been tried out in the case studies described in Section 4, whereas steps 3 and 4 are only initial suggestions that have not yet been evaluated. The proposed steps of ACE are:

1. *Assessment of CSPs from process descriptions or practice.* Assess CSPs with respect to process descriptions or actual development. So far, we have only made assessments based on process descriptions and a three-grade scale (see Section 4), but assessments based on real practice and more fine-grained scales are of course also possible.
2. *Adding colours to the anatomy.* Fill each box of the anatomy with a colour that represents the grade that the CSP in scope has been assigned in the context. Figure 2 displays the assessments from case C2 of the case studies (see Section 4). Black represents grade 3 (CSP fully covered), grey represents grade 2 (CSP partly covered), and white is used to visualize grade 1 (CSP not covered).
3. *Identify improvement areas.* Improvement areas can be identified from the anatomy by visual inspection. The borders between grades are of special interest and one such border, between grade 1 and grade 2, is represented by the dotted line in Figure 2. It shows that SPEC3, TUN3, TUN6, TUN2, VER4, and EST1 are primary candidates to address if the main target is to raise the ability from grade 1 to grade 2.
4. *Construct improvement plan.* It is often the case that not all improvement candidates can be addressed. We suggest that the anatomy can aid in the prioritization and that CSPs that are important for supporting other CSPs and/or blocks the possibility to move "higher" should be prioritized. In Figure 2, this should result in EST1 and VER4 as top candidates for improvement, which we also believe is conformant with intuition when considering the shape of the dotted line. An option would be to also add TUN2 and SPEC3 to carry out improvement efforts in all CSPAs. Since the CSPAs correspond well to roles this means that exactly one improvement effort is assigned to each role.

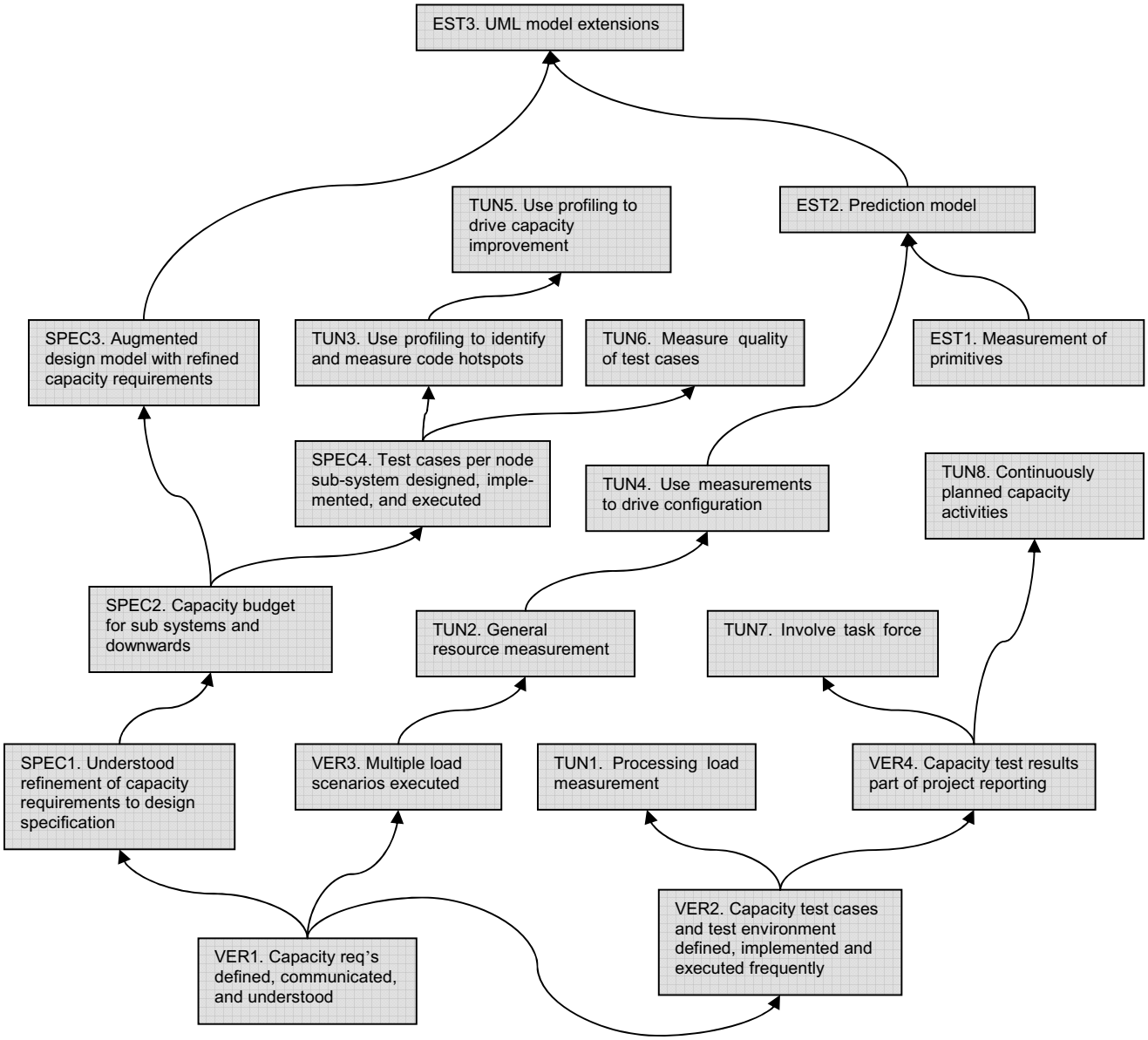


Figure 1: The proposed anatomy of capacity engineering

Table 2: Case descriptions

ID	Case type	System type	Organization size	Product lifecycle	Release
C1	Interviews	Several	Varying	Varying	Varying
C2	Development organization	Radio Base Station	< 500	Initial	0
C3	Development organization	Operation, Administration, Maintenance and Provisioning	> 2000	Maintenance	> 15
C4	Development organization	Multimedia Systems	< 500	Development	< 5
C5	Process	-	-	-	-

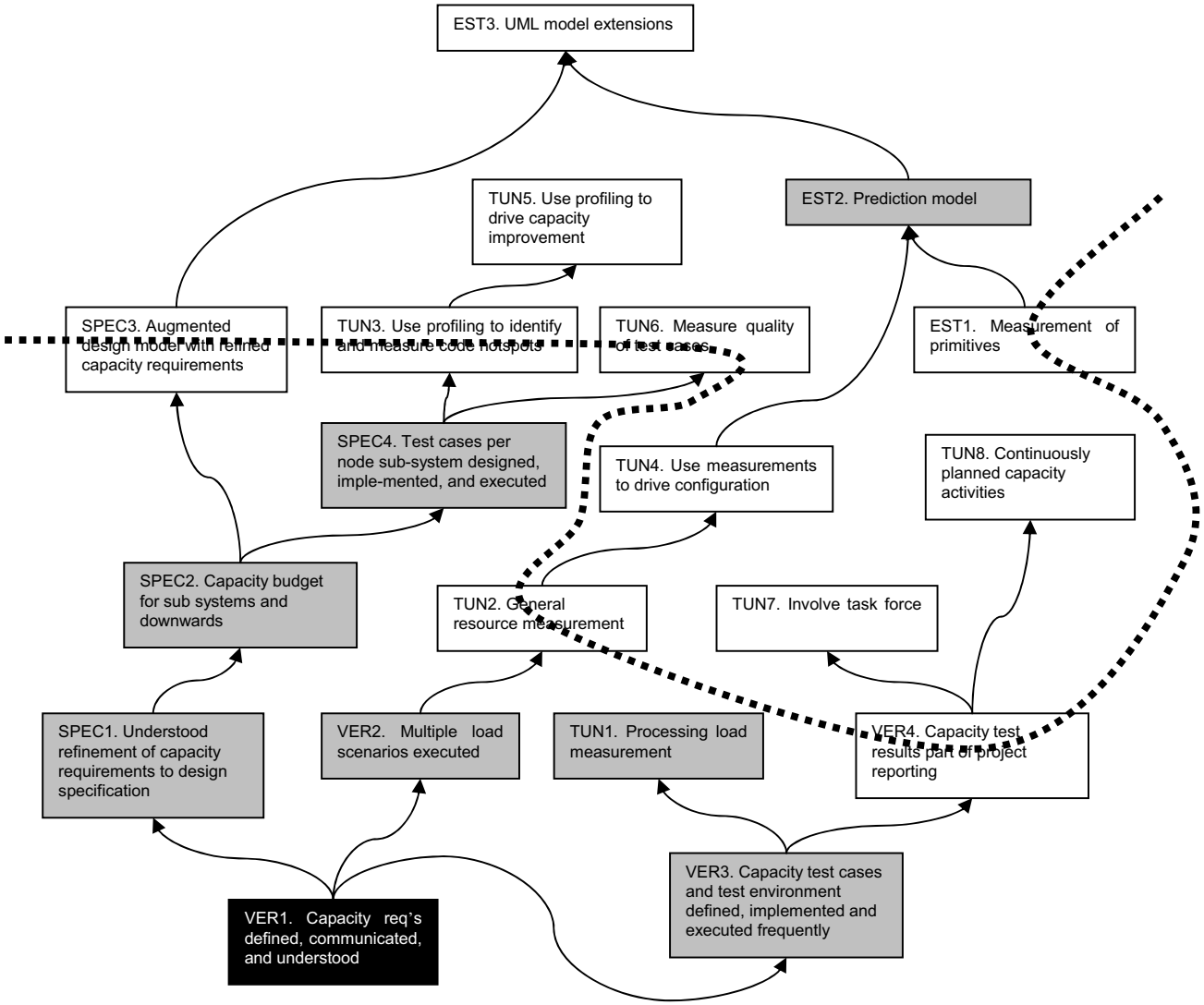


Figure 2: Identification of improvement candidates using the anatomy with assessments from C2

4. THE CASE STUDIES

The CSP assessment has been tried out in three case studies involving five cases in total, C1-C5 (see Table 2), from the perspective of process improvement:

- C1 is an assessment of the interview results [1] from which the anatomy has been developed. Thus, C1 does not represent a single development process but instead the actual frequency of use in ten separate Ericsson organizations at four sites.
- C2-C4 each represent separate development organizations within Ericsson (C2 and C3 were also part of the interview series that constitutes C1).
- C5 is an assessment of the OpenUP/Basic process [8]. The development of a capacity method plug-in to OpenUP/Basic is described in [2].

For each of the cases C2-C5 the process in scope has been reviewed and assessed with respect to the CSPs. Thus, we have looked for activities similar to the CSPs of the anatomy in each of the cases. In other words, for each case the presence of each single CSP has been rated 1 (not covered), 2 (partly covered), or 3 (fully covered), and the results are shown in Table 3 and in Figure 3-Figure 7 below. C1 is special in the sense that it does not represent a single process. Instead, the assessment is based on the results of the interview series involving several development organizations with different development processes. Even though the application of the anatomy is primarily relevant for cases C2-C5, the presence of C1, and also C5, is interesting for comparison reasons.

Table 3: Case study results

ID	C1	C2	C3	C4	C5
EST1	2	1	1	2	1
EST2	1	2	2	1	1
EST3	1	1	1	1	1
SPEC1	2	2	2	2	2
SPEC2	2	2	2	1	1
SPEC3	1	1	1	1	1
SPEC4	1	2	3	2	1
TUN1	3	2	2	1	2
TUN2	2	1	2	1	1
TUN3	2	1	2	1	1
TUN4	1	1	1	1	1
TUN5	1	1	2	1	1
TUN6	1	1	2	1	1
TUN7	1	1	2	1	1
TUN8	1	1	2	1	1
VER1	3	3	3	2	3
VER2	2	2	3	2	2
VER3	2	2	3	1	1
VER4	1	1	2	1	2
Mean:	1.58	1.47	2.00	1.26	1.32

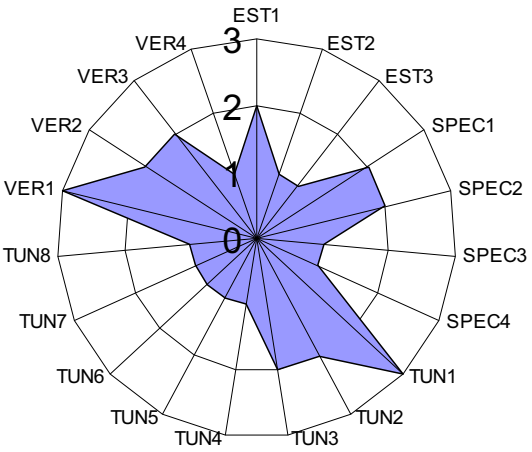


Figure 3: C1 results

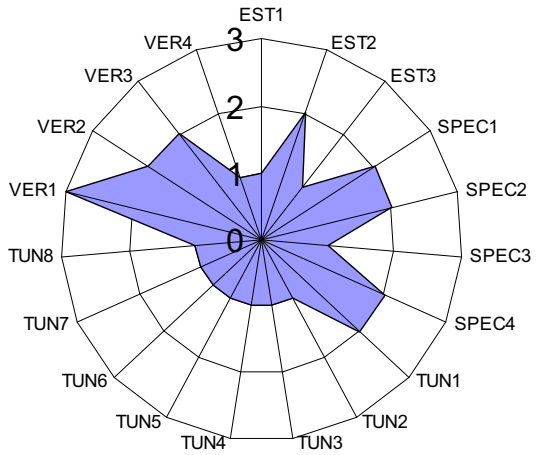


Figure 4: C2 results

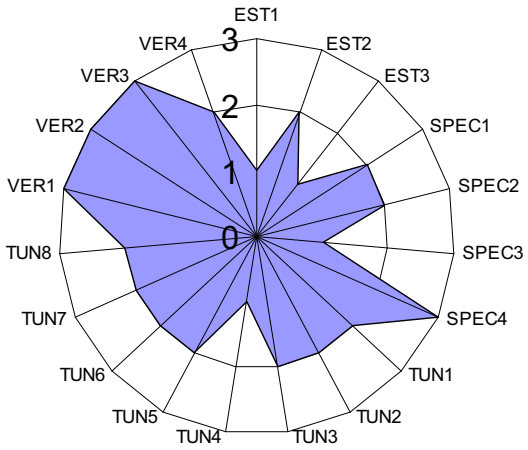


Figure 5: C3 results

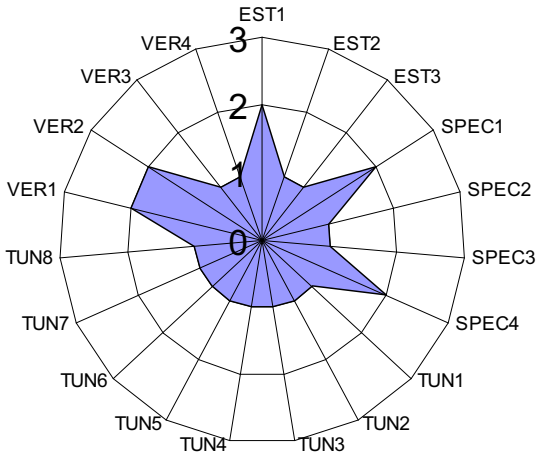


Figure 6: C4 results

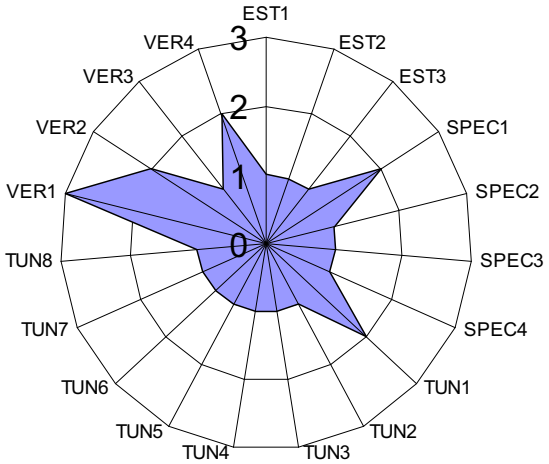


Figure 7: C5 results

5. ANALYSIS AND DISCUSSION

An overall observation from Figure 3-Figure 7 is that all cases tend to reach their peaks in the "beginning" of each CSPA. Consider, for instance, C1 (see Figure 3) that displays its best grades in the initial parts of each CSPA and reaches its peaks for TUN1 and VER1. This is, of course, an expected outcome since CSPs are generally arranged in the order of increasing maturity within each CSPA (especially expected in the C1 case since it is the interview series from which the anatomy has been deduced). The anatomy (see Figure 1) also shows that most initial CSPs need to be fulfilled along the path to gain most benefit of the more advanced CSPs.

Another general observation is that VER1-2 are rated highest in almost all cases. These CSPs focus on the ability to verify the capacity requirements. The reasons for this is quite natural: First, it is not sensible to specify requirements that cannot be tested, and second, testing of capacity has to be prepared well in advance, so there is a strong pull for directives from testers. Compared to VER1-2, SPEC1-2 has not received attention to the same extent. SPEC1-2 is about communicating the specification to design, which should be as important as communication with testing. One can only speculate about the reason for this deviation. We like to think that the transition requirements-design is a larger semantic challenge than the transition requirements-testing if the same levels of abstractions are used in specification and testing. Thus, the process requirements-design is more a matter of negotiation and creativity.

C2 (see Figure 4) is very similar to C1. The major difference is that C1 is stronger on Verification. However, due to the recent adaptation of the development process to cover the prediction facilities of SPE [9], the EST2 CSP ("Prediction model") of C2 fulfils the criteria for grade 3 if considered in isolation. Nevertheless, it has only been assessed grade 2 due to the CSP's relationship with TUN4 ("Use measurements to drive configuration"). The reason is that the measurements of TUN4 are needed to update the prediction model and make it useful in the next development step. It is interesting to notice, though, that this is not the C2 organization's "fault" since measurements need to be provided from another organization (namely, the platform on which C2's application is built). This circumstance further highlights that capacity in most cases is achieved by contributions from different components.

C3 is the case with the best rating (an average fulfilment of grade 2, see Table 2), which is clearly visible from comparing the chart surface displayed in Figure 5 with the chart surfaces of the other cases. The big differences are the Verification and Measurement and Tuning CSPAs, for which C3 demonstrates a process that is considerably better adapted to capacity than the other cases' processes are. The C3 process is particularly strong regarding the Verification CSPA; only the grade 2 on VER4 ("Capacity test results part of project reporting") prevents it from reaching complete fulfilment in this area. Moreover, raising the grade from 2 to 3 on VER4 should be a relatively easy improvement to accomplish.

All cases show a week grading in estimation, even in C3 with the most mature products. In this case this might be explained by the recent process re-engineering that probably had a higher focus on recent hot topics than those of the inception of the product. For some applications the first release was in the mid-1990's.

In contrast to C3, the C4 case (see Figure 6) has the process that leaves most room for capacity adaptation, and it is reasonable to believe that a major explanation is that the product is relatively new. No CSP of the case has been assessed the grade 3, but like the other cases the peaks of the C4 process are reached for the initial CSPs of each CSPA.

C5 (see Figure 7), the OpenUP/Basic Process [8] shows the properties that the "Basic" part of the name implies: The process covers the basic parts of all CSPAs except Estimation and Prediction, for which it has no support at all. However, these are expected results since the process is intended to be minimal. It is also extensible and as already mentioned there is current work with creating a capacity method plug-in to OpenUP/Basic [2].

Based on the observations and discussion above we believe that our anatomy is verified in the sense that the order between the different CSPs is mainly followed by our case studies. The explanations of confirmed and unexpected results are made in terms of the maturity of the product and the most emphasized life cycle. Thus, we believe that an anatomy and a list of CSPs need to be accompanied with information of what processes to consider. In terms of the radar charts in Figure 3-Figure 7 we expect a spiral pattern in clock-wise direction.

A typical organization in initial stage with a new product starts with the first CSP in each of the CSPAs in the order EST-SPEC-TUN-VER. The second round of improvements both increases the grade of the first CSPs as well as starting growth of sub-sequent CSPs in each area.

An organization in developing stages is more likely to make good use of our anatomy and accompanying method by being able to create a stepwise, customized improvement plan based on achievements so far.

A mature organization with a mature product is represented in C3 in our cases. Such an organization is well off in many CSPAs, but can use the anatomy in updating earlier processes, such as initial estimations. Further, this organization can take the challenge to create models and processes for bridging the semantics between requirements and design. Given the routine maintenance work of such an organization there is an economic incentive in systematizing this.

6. CONCLUSIONS AND FUTURE WORK

Earlier research has derived empirically grounded capacity sub-processes (CSPs) that are necessary to improve for organizations committed to fulfil long-term requirements on capacity. In this paper we have placed the CSPs in an anatomy that suggests in what order an organization can naturally address the different CSPs. This anatomy has been used to describe the development of CSPs in three different case studies involving five cases. Our conclusion is that if the anatomy is complemented with information of the maturity of the organization and the product, then the anatomy is a useful tool for describing real-world, successful process initiatives. If this also can be transferred to normative knowledge is still a question for future work.

7. ACKNOWLEDGEMENTS

We thank the participating design organizations for generous access to their processes. This project was funded by Ericsson AB, the Swedish Foundation for Strategic Research, and Vinnova.

8. REFERENCES

[1] Borg, A., M. Patel, and K. Sandahl. "Good Practice and Improvement Model of Handling Capacity Requirements of Large Telecommunication Systems", in the *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06*), pp. 245-250, Minneapolis, USA, Sept 11-15, 2006

[2] Borg, A., M. Patel, and K. Sandahl. "Integrating an Improvement Model of Handling Capacity Requirements with OpenUP/Basic Process", *Proceedings of the International working conference on Requirements Engineering: Foundations for Software Quality (REFSQ'07)*, Trondheim, Norway, June 11-12, 2007.

[3] Chung, L., B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.

[4] Davis, A. M. *Software Requirements: Objects, Functions and States*, Prentice Hall, Upper Saddle River, NJ, 1993.

[5] http://www.eclipse.org/epf/, Eclipse Process Framework Project (EPF), accessed 13 September 2007.

[6] Kruchten, P. *The Rational Unified Process: An Introduction*, Second ed., Addison-Wesley, Reading, MA, 2000.

[7] Moe, J., and D. Carr. “Using Execution Trace Data to Improve Distributed Systems”, *Software—Practice & Experience*, vol. 32, no 9, pp. 889-906, July 2002.

[8] http://www.eclipse.org/epf/openup_component/openup_index.php, OpenUP Component, accessed 13 September 2007.

[9] Smith, C.U., and L.G. Williams. *Performance Solutions. A Practical Guide to Creating Responsive, Scaleable Software*, Addison-Wesley, 2002.

[10] http://www.omg.org/technology/documents/formal/schedulability.htm, UML Profile for Scedulability, Performance, and Time, accessed 13 September 2007.

Evaluating Software Evolvability

Hongyu Pei Breivold
ABB Corporate Research
721 78 Västerås, Sweden
+46 21 323243
hongyu.pei-breivold@se.abb.com

Ivica Crnkovic
Mälardalen University
721 23 Västerås, Sweden
+46 21 103183
ivica.crnkovic@mdh.se

Peter Eriksson
ABB AB
721 78 Västerås, Sweden
+46 21 344310
peter.j.eriksson@se.abb.com

ABSTRACT
Software evolution is characterized by inevitable changes of software and increasing software complexities, which in turn may lead to huge cost unless rigorously taking into account change accommodations. This has intensified the need on evolvable software systems that can correspond to changes in a cost-effective way. Nevertheless, although software evolvability is one of the most important quality attributes of software, it is not precisely defined today. Besides, the lack of evolvability model hinders us from analyzing, evaluating and comparing software systems in terms of evolvability. To address these issues, we distinguish software evolvability from maintainability in this paper and outline a suggestion for an evolvability model which analyzes software evolvability from various perspectives, as well as an evolvability evaluation method. The model and the method are evaluated through its application in an industrial automation system. The contribution of this paper is the initial establishment of an explicit definition of software evolvability, an evolvability model and an evolvability evaluation method that can be applied for large complex software-intensive systems.

Keywords
Software evolvability, maintainability, quality model

1. INTRODUCTION
Software maintenance and evolution are characterised by their huge cost and slow speed of implementation [3]. The ability to change and evolve software quickly and reliably has become a challenging issue for both software engineering community and industry.

Industry rarely develops new products from scratch [11]. New features, constraints and enhancements of most new products are usually built on top of the earlier versions of software products. This is due to the fact that in most cases, the cost of evolving software is lower than developing from scratch [20]. Typical examples are industrial automation systems. Since industrial automation systems are often long-lived software-intensive systems that can have a lifetime of 20-30 years, they are subject to changes and may undergo a substantial amount of modifications in order to be responsive to the constantly changing demands from the marketplace, stakeholders, business requirements, environment or technologies during their lifecycles. This implies that these software-intensive systems become more and more complex and may contain up to several million lines of code as the software is enhanced, modified and adapted during the software evolution process. Complexity increases unless work is done to maintain or reduce it [15]. These phenomena in continuing change and increasing complexity were recognized by Lehman and expressed in his well-known laws of software evolution [15]. The properties of large software systems noted by F. P. Brooks [6], e.g. software complexity, inevitable changes of software systems and invisibility in terms of software structure representation, further confirm the software evolution characteristics and exhibit the intensified need on evolvable software systems that can be long-lived and correspond to changes in a cost-effective way.

One way to ensure that any software system does not deteriorate as it is evolved is to provide feedback to the development team about the evolvability. Statistics have shown that the largest part of lifecycle costs for long-lived software systems is concerned with the evolution of the software [2] to cope with the challenges of the continuing change, increasing complexity and the tendency of declining software quality. Therefore, the systems' capability to cost-effectively adapt to and accommodate various changes has become essential for companies to survive in the competition and maintain a leading position among competitors. The inability to effectively and reliably evolve software systems means loss of business opportunities [3]. Consequently, there is strong demand to carry out software evolution efficiently and reliably, thus, to prolong the productive life of a software system.

Today, software needs to be changed on a constant basis with major enhancements within short timescale, through coping with the changing environments and the radically changing requirements. All these put critical demands on the software system's capability of rapid modification and enhancement. In this sense, software evolution is one term that can express the software changes during software system's lifecycle and software evolvability is an attribute that describes the software system's capability to accommodate these changes with the condition of having the lifecycle costs under control. As software evolution activities are performed, essential characteristic software evolvability must be considered. Nevertheless, although software evolvability is one of the most important quality attributes or characteristics of software, it is not precisely defined today. It is not explicitly defined in any well-known quality models that we have investigated, e.g. McCall's quality model, ISO/IEC 9126, etc. Because of the lack of a standard definition, many people use software evolvability as synonymous to software maintainability. Although both have similarities in many senses, software maintainability and evolvability have specific focus, which has resulted in confusion in understanding and applying similar concepts designated differently. Furthermore, software evolvability is affected by many factors and it is difficult to quantify.

Thus, in this paper, we intend to (i) show differences between software maintainability and evolvability, (ii) define a software evolvability model, (iii) identify the required subcharacteristics of software evolvability based on the analyses of several well-known quality models and comparisons between evolvability and maintainability, and (iv) evolvability evaluation method. This evolvability model is established as a first step towards quantifying evolvability, a base and check points for evolvability analysis and evaluation as well as evolvability improvement. Further we demonstrate the model and the method through an industrial case study.

The rest of the paper is structured as follows. Section 2 analyzes several existing well-known quality models, compares evolvability with maintainability and gives a definition of software evolvability and proposes the evolvability model. Section 3 presents evaluation of software evolvability using the model and relates it to different architecture evaluation methods that may be adapted for evolvability evaluation. A comparison between the evolvability model and the related methods is also addressed in this section. Section 4 presents a case study in applying the evolvability model and evaluation method. Section 5 concludes the paper and outlines the future work.

2. SOFTWARE EVOLVABILITY MODEL

To be able to define the evolvability model we start with a short analysis of different quality models in which we can find the elements of evolvability. In particular we analyze subcharacteristics of maintainability and defined the subcharacteristics of evolvability. Based on this analysis we provide the evolvability model.

2.1 Analysis of Quality Models

A quality model provides a framework for quality assessment. It aims at describing complex quality criteria through breaking them down into concrete subcharacteristics. The best known quality models include McCall [17], Boehm [4], FURPS [18], ISO 9126 [13] and Dromey [10]. The quality characteristics that are addressed in these quality models are summarized in Table 1. As shown in Table 1, although several quality attributes are correlated to software evolvability, e.g. adaptability, extensibility and maintainability, the term evolvability is not explicitly addressed in either of the quality models. On the other hand, this table provides useful inputs for the establishment of the software evolvability model, e.g. the identification of subcharacteristics of evolvability.

2.2 Evolvability

We define software evolvability as follows:

Definition: *Software evolvability is the ability of a software system to adjust to change stimuli, i.e. changes in requirements and technologies that may have impact on the software system in terms of software structural and/or functional enhancements, while still taking the architectural integrity into consideration*

Software evolvability is both a business issue as well as a technical issue, since the stimuli of changes can come from both perspectives, including change of business models and business objectives, changes in environment, quality requirements, functional requirements, underlying technologies as well as emerging technologies.

Since maintainability is covered in most of the well-known quality models and it is generally considered as most related to evolvability, we will study the definitions of maintainability in order to make the definition and features of evolvability distinguishable. A summary of the definitions of maintainability in various quality models is presented in Table 2.

Table 1 Quality characteristics addressed in quality models

Quality Characteristics	McCall	Boehm	FURPS	ISO 9126	Dromey
Adaptability			x Supportability	x Portability	
Compatibility			x Supportability		
Correctness	x				
Efficiency	x	x		x	x
Extensibility			x Supportability		
Flexibility	x				
Human Engineering		x			
Integrity	x				
Interoperability	x			x Functionality	
Maintainability	x	x	x Supportability	x	x
Modifiability		x		x Maintainability	
Performance			x		
Portability	x	x		x	x
Reliability	x	x	x	x	x
Reusability	x				x
Supportability			x		
Testability	x	x		x Maintainability	
Understand-ability		x		x Usability	
Usability	x		x	x	x

Table 2 Definitions of maintainability in quality models

Quality Models	Maintainability Definition	Focus
McCall	The effort required to locate and fix a fault in the program within its operating environment	Corrective maintenance
Boehm	It is concerned with how easy it is to understand, modify and test.	Understandability, modifiability and testability
FURPS	Implicit	Adaptability, extensibility
ISO 9126	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.	Analyzability, changeability, stability, testability

We intend to distinguish software evolvability from maintainability from a collection of aspects that characterize them, such as software change stimuli that trigger the changes, type of change, impact on development process and type of scenarios used in analysis, etc. The differences are summarized in Table 3.

Table 3 Comparisons between evolvability and maintainability

Characteristics	Evolvability	Maintainability
Software Change Stimuli	Business model, business objectives, functional and quality requirement, environment, underlying and emerging technologies, new standards, new versions of infrastructure	Defects, functional requirement, requirements from customers
Type of Change	Coarse-grained, long term, higher level, [19] radical functional or structural enhancements or adaptations	Fine-grained, short term, localized change [19]
Focus Activity	Cope with changes	Keep the system perform functions [21]
Software Structure	Structural change	Relatively constant
Analysis Scenarios	Growth scenarios (change scenarios)	Existing use case scenarios
Development Process	May require corresponding process changes	Relatively constant
Architecture Integrity	Conformance is required	Conformance is preserved

2.3 Software Evolvability Model

Since software evolvability is a multifaceted quality attribute, we propose a software evolvability model with identification of the required subcharacteristics that a software system needs to possess in order to easily adapt to various changes during software evolution. The subcharacteristics that are identified and selected for the evolvability model are based on their importance for software developing organizations in general and their relevance for evolving software in a cost-effective way.

The process of identifying and selecting subcharacteristics is based on the earlier mentioned maintainability and evolvability analysis as well as the mentioned quality models. Evolvability-related subcharacteristics are identified and classified into six aspects. This classification is based on all the quality characteristics that are covered in the previously mentioned quality models. Each aspect addresses a set of quality characteristics that are covered in the well-known quality models as illustrated in Table 4. Besides, we have followed ISO 9126 standards and checked their quality attributes against our classification for completeness. Apart from the development quality attributes that are explicitly addressed in the evolvability model, the operational quality attributes, such as performance, reliability are also indirectly addressed in the sense that the improvement of these attributes are handled through e.g. analyzability and changeability. Portability and extensibility are explicit in the classification because they are essential for software evolvability. One may argue that extensibility and changeability are closely related with each other. The reason we make extensibility explicit is that additional feature enhancement is one of the essential activities in software evolution. As a result, these identified subcharacteristics are relevant for evolution of software-intensive systems and cover the ranges of potential future changes that a software system may encounter during its life cycle.

Table 4 Classifications of Evolvability-Related Subcharacteristics

Classification	Quality Characteristics in Quality Models
Analyzability	Human Engineering, Understandability
Changeability	Flexibility, Modifiability
Integrity	Reusability
Extensibility	Extensibility
Portability	Adaptability, Compatibility, Interoperability
Testability	Correctness, Efficiency

The proposed evolvability model provides a base and a catalog of check points for analyzing and evaluating software evolvability. The subcharacteristics that evolvability incorporates and their motivations are explained below.

Analyzability The capability of the software system to enable the identification of influenced parts due to change stimuli (adapted from [13]). The change stimuli include changes in business model, business objectives, functional and quality requirements, environment, underlying technologies and emerging technologies, new standards, new infrastructure, etc.

Analyzability is important since a software system must have the capability to be analyzed and explored in terms of the impact to

the software by introducing a change. Many perspectives can be included in analyzability dimension, e.g. decisions on what to modify, analysis and exploration of emerging technologies from maintenance and evolution perspective, etc.

Integrity The capability of the software system to maintain architectural coherence while accommodating changes.

Integrity is a key element that may be easily ignored during software evolution. It is mostly related to understanding and coherence to the previous architectural decisions and adherence to the original architectural styles, architectural patterns or strategies. Insufficient understanding of the initial architectural constructs may have indirectly negative consequences on software structures and lead to evolvability degradation in the long run. However, taking integrity as one subcharacteristic of evolvability does not mean that the architectural constructs are not allowed to be changed. On the contrary, it helps in recognition, extraction and documentation of these architectural constructs as well as prevents unconscious violations against architectural principles. As a result, any necessary changes to the architecture can be conducted in a controlled way. The software architecture of an evolvable software system should allow considerable unanticipated changes in the software without compromising system integrity and invariants and can evolve in a controlled way [3].

Changeability The capability of the software system to enable a specified modification to be implemented [13].

Changeability is important since a software system must have the ease and capability to be changed without negative implications to the other parts of the software system or in a controlled way. The changeability of the software should be analyzed in correspondence to various evolution categories, e.g. new version of infrastructure or meeting business objectives. Thus, changeability is correlated to extensibility and portability in the sense that any re-factoring candidates identified in them will be eventually justified through changeability. Changeability is closely related to coupling, cohesion, modularity and software complexity in terms of software design and coding structure [14], though it is often constrained by business and economical factors.

Portability The capability of the software system to be transferred from one environment to another [13]. Portability is an example of a property that is not a subcharacteristic of maintenance but it is essential for evolvability.

Portability is one important characteristic for long term development due to the rapid technical development on hardware and software technologies. It is concerned with hardware and/or software changes, including interface and platform aspects. Therefore, it is one of the key enablers that can provide possibility to choose between different hardware and operating system vendors as well as various versions of frameworks. Portability analyses need to be made from evolution perspective, e.g. exploration of emerging technologies that may affect portability, analyzing the effect on the software architecture in terms of portability, etc.

Extensibility The capability of the software system to enable the implementation of extensions to expand or enhance the system with new capabilities and features with minimal impact to existing system. Extensibility is a system design principle where the implementation takes into consideration of future growth.

Extensibility is important since a software system must have the ease and capability to add on extra functionality and features, extra components and services to keep up with the plethora of standards, customer requirements, market requirements, etc. In order to keep its competitive edge, a software system must constantly raise the service level through supporting more functionality and providing more features [5]. This property is also characteristic for evolvability, but not for maintainability.

Testability The capability of the software system to enable modified software to be validated [13].

Testability is concerned with the verification of a software system since software modification may lead to errors and side effects, e.g. changes to one part of a system may have an unintended effect on another part of the system. Therefore, every step in the transformation and changes of software constructs need to be tested. Test cases that cover both the original and emerged changing requirements need to be identified to ensure that the system still can fulfill the original requirements and perform its intended function while meeting the new requirements.

From the list of the subcharacteristics we could assume that maintainability is a subset of evolvability, but this is only partially true. Evolvability and maintainability have different goals (changes vs. preservation as explained in Table 3) and the subcharacteristics will be evaluated in relation to these goals.

Analyzability and integrity are the center subcharacteristics and base for evolvability evaluation. The reason is that analyzability is the first core step to identify the influenced parts due to change stimuli; whereas integrity investigation helps to gain comprehensive understanding of architectural constructs related to other evolvability subcharacteristics of the software system, such as changeability, extensibility, portability and testability, so as to guarantee that any re-factorings made to the system will be well-planned instead of unconsciously violating existing reasonable architectural decisions.

During the software evolution process, there may be shifted focus among portability and extensibility depending on the types of emerging changes. Nevertheless, analyzability, changeability, testability and integrity are the main subcharacteristics that are required in all circumstances.

3. EVALUATING EVOLVABILITY

Software evolution and software evolvability can be examined in different phases of systems lifecycle, e.g. requirement phase, architectural phase, detailed design, and implementation and integration phases [9]. In this paper, we focus on assessing software evolvability at architectural phase. This is because software architecture is a key asset in software systems and it has tight connection to the system's quality requirements in the sense that software architectures allow or preclude nearly all of the system's quality attributes, or vice versa, the quality attributes of a software system are determined by its architecture [8].

3.1 Evaluation Method Supporting the Evolvability Model

In order to address the evolvability subcharacteristics systematically, we have generalized an approach for evolvability evaluation from an industrial case study. The application of this method and the evolvability model will be examplified in more

details in a case study in section 4. The approach comprises two phases.

Phase 1: Analyze the implications of change stimuli on software architecture

This phase addresses analyzability subcharacteristics as shown in Figure 1, and includes the following two steps:

Step 1: Identify requirements on the software architecture

Step 2: Prioritize requirements on the software architecture

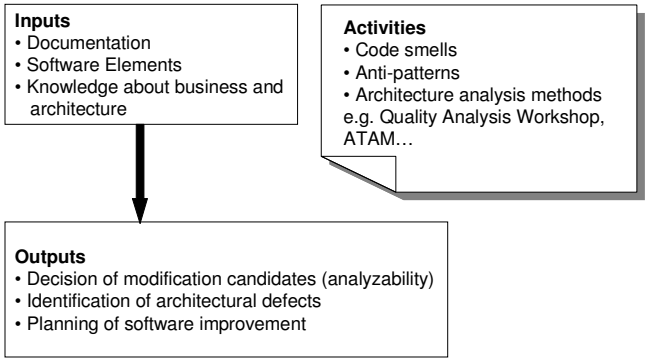


Figure 1 Software Analysis Process (Phase 1)

Phase 2: Analyze and prepare the software architecture to accommodate change stimuli and potential future changes

This phase addresses integrity, changeability, extensibility, portability and testability subcharacteristics as shown in Figure 2, and includes the following steps:

Step 3: Extract architectural constructs related to the identified issues from phase 1

Step 4: Identify re-factoring components for each identified issue

Step 5: Identify and assess potential re-factoring solutions from technical and business perspectives

Step 6: Identify and define test cases

Step 7: Present analysis results

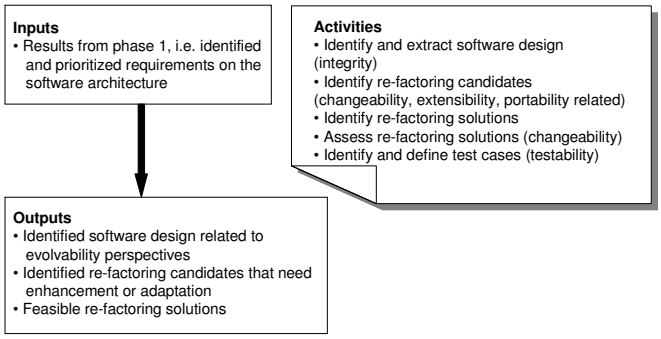


Figure 2 Software Improvement Process (Phase 2)

To summarize, the outputs of software evolvability evaluation include (i) Identified and prioritized requirements on the software architecture (ii) Established base for common understanding of these requirements from stakeholders within organizations (iii) Identified re-factoring candidates that need enhancement or adaptation (iv) Feasible re-factoring solutions and (v) identification of test cases.

3.2 Other Methods

There exist many architecture evaluation methods today. Some of them may be adapted to analyze software evolvability. Following is a brief description of these methods.

ATAM The Architecture Tradeoff Analysis Method (ATAM) [8] is a method for evaluating software architectures in terms of quality attribute requirements. It is used to expose the risks, non-risks, sensitivity points and trade-off points in the software architecture, therefore to achieve better architecture. It aims at different quality attributes and supports evaluation of new types of quality attributes.

SAAM The Scenario-based Architecture Analysis Method (SAAM) was originally created for evaluating modifiability of software architecture. The main outputs from a SAAM evaluation include a mapping between the architecture and the scenarios that represent possible future changes to the system, which provides indications of potential future complexity parts in the software and estimated amount of work related to the changes.

ALMA The Architecture Level Modifiability Analysis [1] is a method for analyzing modifiability based on scenarios. The outputs from an ALMA evaluation include maintenance prediction to estimate required effort for system modification to accommodate future changes, risk assessment to identify the types of changes that the system shows inability to adapt to, and software architecture comparison for optimal candidate architecture.

EBAE Empirically-Based Architecture Evaluation [16] defines a process for defining and using a number of architectural metrics to evaluate and compare different versions of architectures in terms of maintainability.

ABAS Attribute-Based Architectural Styles [4] build on architectural styles by explicitly associating with reasoning frameworks, which are based on quality attribute-specific models.

3.3 Correlations among Evaluation Methods

Among the related evaluation methods, ALMA and SAAM focus more on modifiability (changeability), EBAE on maintainability using metrics such as coupling, size and complexity, and ATAM supports multiple attributes. Since software evolvability is a multifaceted attribute, incorporating changeability among other subcharacteristics, ALMA, SAAM and EBAE will not be sufficient enough to evaluate software evolvability. Regarding ATAM, although it can support multiple quality attributes, it has one liability in dealing with future changes due to the limitation of the scenario generation process, since some evolvability scenarios may be missed which may result in wrong judgments about the current architecture [7].

The software evolvability model that we have outlined is appropriate for evolvability analysis because it pinpoints the dimensions that software architects and analysts need to consider in carrying out software evolution activities during the software evolution process. As illustrated in Figure 1, we see also the benefit of using ATAM as a basis for architecture analysis in combination with the evolvability model for evolvability evaluation.

4. CASE STUDY

The application of the proposed software evolvability model and the evaluation method was carried out on a large industrial automation system at ABB. During the long history of product development, several generations of automation controllers have been developed as well as a family of software products, ranging from programming tools to varieties of application software that support every stage of the software system life cycle. The case study was focused on the latest generation of automation system.

4.1 Evaluated System

The software system in the automation controller today has a tremendous huge code base, consisting of several million lines of code with support for a variety of different applications and devices. All the source code is compiled into a monolithic binary software package, which has grown in size and complexity as new features and solutions are added to enhance functionality and to support new hardware, such as devices, I/O boards and production equipment. Besides, the software package also consists of various software applications, aiming for specific tasks that enable the automation controller to handle various applications in painting, arc welding, spot welding, gluing, machine tending or palletizing, etc.

Due to long life of products and continuous improvements of the products, software evolvability is one of the most important properties that is of interest.

4.2 Goals

The aim of the case study was to analyze software architecture of the automation system with respect to its evolvability through applying the software evolvability method. The motivations to this case study came from the emerging critical issues in terms of software evolution, which are:

- How to improve software system quality?
- How to improve the ability to enhance functionality in existing software system?
- How to build new products for dedicated market within short time?
- How to enable the ease and flexibility of distributed development of products?

Of all these questions, the root challenge is how to evaluate software evolvability and analyze whether the software system has the capability to quickly accommodate to changes. This is the necessary step towards improving software evolvability and preparing the software system for potential evolution.

4.3 Applying the Evolvability Evaluation Method

How to evolve the current monolithic automation controller software? Is it possible to evolve the controller software to meet the business objectives? We applied the software evolvability evaluation method and checked against the evolvability model to address theses issues.

Step 1: Identify requirements on the software architecture

Any change stimuli result in a collection of requirements that the software architecture needs to adapt to. The aim of this step is to extract requirements that are essential for enhancing and preparing the software architecture to cost-effectively accommodate change stimuli. Workshops and scenario-based architecture analysis methods can be used for this purpose. In our case study, several workshops were conducted for requirement identification.

The change stimuli in this case study came from the business objectives, i.e. time to market, quality improvement and enabling distributed development process. The main idea to accommodate to the change stimuli was to cope with the monolithic-related issues through developing base software for domain-specific applications to build on. The base software consists of a software kernel which is the mandatory building block for all applications, as well as common extensions which are commonly used by all the applications. The base software can be packaged into software development kit, which provides necessary tools and documentation for application development. The domain-specific application parts will be separated from the base software and any application-specific extensions can be built on top of the base without the need of access to source code. This implies that the base software and domain-specific applications can be developed independently and have separate release cycles. Application developers can work more freely than before without being constrained by the release cycles of the base software. To achieve this, corresponding requirements were identified to enable the migration of monolithic architecture to modular one.

Step 2: Prioritize requirements on the software architecture

All the requirements identified from the first step need to be prioritized. In the case study, the priorities for requirements were ranked into three steps: (i) enable build of existing types of extensions, i.e. to fix all interfaces that prevent from building existing extensions after building the kernel (ii) enable new extensions and simplify interfaces that are difficult to understand and may have negative effects when implementing new extensions (iii) scale kernel.

Step 3: Extract architectural constructs related to the respective identified issue

In this step, we mainly focus on architectural constructs that are related to the previously identified issue. Take portability issue for example, the evaluated system is the latest generation of automation controller software, which is an evolutionary step based on earlier generations. One of the main initial design goals was to make the software portable across different target operating system (OS) platforms, as well as to run it in form of a “Virtual Controller” hosted on a general purpose computer, such as a UNIX workstation or a PC. The architecture style for the current generation automation control software is layered architecture, and within the layers object-oriented architecture. The main enabler for portability is the portability layer in the architecture. The portability layer provides interfaces for application software in the controller, including OS abstraction, POSIX file API, device driver interfaces, basic services and reusable class library. To summarize, this step is necessary to help us understand the system related to the problem issue and to discover any architectural defects around it.

Step 4: Identify re-factoring components for each identified issue

In this step, we identify the components that need re-factoring in order to fulfill the prioritized requirements. For example, in the case study, to achieve the build- and development-independency between kernel and extensions, the low-level basic services were identified as one of the re-factoring components.

Step 5: Identify and assess potential re-factoring solutions from technical and business perspectives

Technical assessment takes into consideration of change propagation and the effect of re-factoring on quality characteristics such as complexity and maintainability of the software. Business assessment estimates the cost and effort on applying re-factoring. In some cases, the solution to a certain re-factoring component is straight forward and we know how to re-factor with local impact. Otherwise, when the implementation is uncertain and may affect several sub-systems or modules, we need to make prototype and investigate the feasibility of potential solutions as well as the estimation of implementation workload.

Step 6: Define test cases

The test cases or scenarios can be defined based on the prioritized requirements on the software architecture. Meanwhile, the software system still needs to fulfill some of the original requirements besides the new required changes. To do this, we need to identify the original test cases as well as the emerging new test cases that cover the affected component, modules or subsystems during the software evolution process. For example, in the case study, we identified test scenarios that enable separation between kernel and extension which are new test cases, and test scenarios for validating if existing domain-specific applications can still work as before without being affected after building the kernel.

Step 7: Present analysis results

The analysis results are transferred to the implementation team for further execution. In fact, the communication between analysis team and implementation team started already during the evaluation process in order to achieve mutual understanding about the re-factoring decisions.

4.4 Analysis

In this case study, we applied the evolvability model to an industrial automation controller and analyzed the software system's evolvability from a collection of dimensions. As stated in [12], software architecture that is capable of accommodating change must be specifically designed for change. Therefore, the application of the evolvability model is a necessary step in analyzing software evolvability and preparing the software system for future changes. The results of the analysis are achieved through applying the evaluation method and are presented as follows.

4.4.1 Analyzability

The knowledge of analyzability is achieved through the first two steps in the evaluation method. In this perspective, we analyze the capability of the software system to enable the identification of influenced parts due to change stimuli. The following lists the most essential activities that were required in the case study for identification of influenced parts due to change stimuli.

(1) Investigate public interfaces This improves both quality and understandability of the current system. It is error-prone to have interfaces defined as public when they should in fact be internal, e.g. application-specific software should not expose public interfaces. All public interfaces should be clearly defined and documented; including the context they can be used. In this way, there will be less and well-defined interfaces, thus to increase software quality and simplify the process of product testing.

(2) Investigate kernel and extensions This provides input to the explicit definition of the scope for kernel**,** common extensions and application-specific extensions.

(3) Investigate build dependencies The separation between kernel and extensions determines that domain-specific applications will always be built last. The build order should start from kernel, common extensions towards application extensions.

(4) Investigate impact on development process The restructuring of the automation controller software will affect the product development processes in the sense that roles, reponsibilities and working procedures, such as product interaction, verification and testing, need to respond to the change stimuli in a corresponding way.

4.4.2 Integrity

The knowledge of integrity is achieved through the third step in the evaluation method. We gained good understanding of the software architecture, although we also discovered minor violations that have taken place on the code level before the actual re-factoring work. This intensified the need of good documentation of architectural constructs and especially rationale behind each design decision.

4.4.3 Changeability

The knowledge of changeability is achieved through step 4 and 5 in the evaluation method. In this perspective, we analyze the capability of the software product to enable a specified modification to be implemented. The underlying assumptions throughout the re-factoring process in this case study were that the applied re-factoring preserves behavior and that the consistency between re-factored artifacts and other software artifacts in the system can be guaranteed, in the sense that requirement specification, architectural design documentation, software code and test specification, etc. should match with each other.

Based on the identified re-factoring components, the respective solution and roadmap for implementation were identified and implementation workload was estimated as well. It became apparent that some modifications were easy to be implemented, while some re-factoring components may lead to considerable change propagation. It is still ongoing work to make comprehensive analysis and judgment of potential alternative solutions.

4.4.4 Extensibility

In this perspective, we analyze the capability of the software system to enhance the system with new functions and features. In the case study, it was desired that domain-specific application developers can create their own application extensions on top of the kernel software in order to respond quickly to market requirements and get rid of the tight constraints from the release cycles of the automation controller software. Therefore, the system is being prepared through executing step 4 and 5. Meanwhile, it became clear that training is necessary so that the domain-specific application developers can easily create their own applications.

4.4.5 Portability

In this perspective, we analyze the capability of the software system to be transferred in case of environment change. The portability issues in this case study include portability analysis

across various target operating system platforms and portability analysis across hardware platform, thus to prepare the software system for potential environment change. It is still an on-going project around this issue, but so far, we have discovered some aspects that need to be addressed, e.g. training for software developers in writing code that enables portability, documentation of guidelines/rules and code examples, proper use of conditional compilation in case of environment switches, etc.

4.4.6 Testability

In this perspective, we validate if the modified software system can still fulfill the original requirements as well as the new required changes. To do this, we identified emerging new test cases that cover the affected component, modules or subsystems as well as the original test cases that the software still needs to fulfill. The possibility of being able to run the program on virtual controllers simplifies a lot for testing.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a software evolvability model and an evolvability evaluation method. We contend that the evolvability of a software system can be analyzed in terms of a collection of subcharacteristics. This evolvability model is established through a systematic analysis of several existing well-known quality models and comparison analysis of distinguishable characteristics between software evolvability and maintainability. We have shown how the evolvability evaluation method and evolvability model can be applied into complex industrial context through a case study, which revealed the structured way of evaluating evolvability as well as the feasibility of using the proposed evolvability model as base and check points when evolving a software system.

Future work remains to be done to further establish the evolvability model to a hierarchical one; we need to further derive the identified subcharacteristics of evolvability to the extent when we are able to quantify them and/or make appropriate reasoning of the quality of service that a software system provides in terms of various sub-characteristic. We need to provide a catalog of guidelines and checkpoints for each sub-characteristic that can be applied in conducting evolvability analysis. We also need to analyze the correlations among the subcharacteristics with respect to constraints and trade-offs. Further we plan to establish a process framework which will enable a consistence analysis when analyzing different subcharacteristics, and when analyzing the evolvability in different phases of the product lifecycle.

6. REFERENCES

[1] Bengtsson. P. O. Architecture-Level Modifiability Analysis. Ph.D Thesis, Blekinge Institute of Technology, 2002.

[2] Bennett, K. Software Evolution: Past, Present and Future. Information and Software Technology 38 (1996) 673-680.

[3] Bennett, K. and Rajlich, V. Software Maintenance and Evolution: a Roadmap. 2000.

[4] Boehm, B. W. et al. Characteristics of Software Quality. Amsterdam, North-Holland, 1978.

[5] Bosch, J. Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach. Addison-Wesley. 2000.

[6] Brooks, F. P. No Silver Bullet. IEEE Computer, Vol. 20, No. 4, 1987.

[7] Ciraci, S. and Broek. P. Evolvability as a Quality Attribute of Software Architectures. 2003.

[8] Clements, P., Kazman, R. and Klein, M. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley. 2002.

[9] Cook, S., Ji, H. and Harrison, R. Dynamic and Static Views of Software Evolution. Proceedings IEEE International Conference on Software Maintenance ICSM, 2001.

[10] Dromey, G. Cornering the Chimera. IEEE Software (January): 33-43, 1996.

[11] Graaf, B. Maintainability through Architecture Development. EWSA, LNCS 3047, pp. 206-211, 2004.

[12] Isaac, D., McConaughy, G. The Role of Architecture and Evolutionary Development in Accommodating Change. Proc. NCOSE'94, 1994.

[13] ISO/IEC 9126-1. International Standard. Software Engineering – Product Quality – Part 1: Quality Model, 2001.

[14] ISO/IEC 9126-3. International Standard. Software Engineering – Product Quality – Part 3: Internal Metrics, 2003.

[15] Lehman, M. Laws of Software Evolution Revisited. Software Process Technology, 5th European Workshop EWSPT, 1996.

[16] McCall, J. A., Richards, P. K. and Walters, G. F. Factors in Software Quality. National Technical Information Service, 1977.

[17] Ortega, M, et al. Construction of a Systemic Quality Model for Evaluating a Software Product. Software Quality Journel, v11, n3, p219-42, Sept 2003.

[18] Pfleeger, S. L. The Nature of System Change. IEEE Software, 1998.

[19] Weiderman. N. H. et al. Approaches to Legacy Systems Evolution. Technical Report CMU/SEI-97-TR-014, 1997.

[20] Yang, H. and Ward, M. Successful Evolution of Software Systems. Artech House Publishers, London, 2003

[21] Loomes, M. J., Nehaniv, C. L. and Wernick, P. The Naming of Systems and Software Evolvability. IEEE Workshops on Software Evolvability 2005.