Quality Improvement in Software Platform Development

Enrico Johansson

LUND UNIVERSITY
Department of Communication Systems
Lund Institute of Technology

ISSN 1101-3931
ISRN LUTEDX/TETS--1053--SE+112P
© Enrico Johansson

Printed in Sweden
KF-Sigma
Lund 2002

This thesis is submitted to Research Board II - Physics, Informatics, Mathematics and Electrical Engineering (FIME) - at Lund Institute of Technology (LTH), Lund University, in partial fulfilment of the requirements for the degree of Licentiate in Engineering.

Contact Information:

Enrico Johansson
Department of Communication Systems
Lund University, Lund
Sweden

email:enrico.johansson@telecom.lth.se

Abstract

A major problem when using software platforms to produce a variety of products relates to keeping a high quality of the platform throughout the development of the products. For a software platform development to be successful, it is essential to master the quality issues when managing and designing the platform.

This thesis presents approaches that enable efficient use of the software platform when it is used as one of the core assets of a product line. Different approaches that are believed to improve the quality of the platform are presented. In order to study the approaches' effect on the development process it is vital to understand what quality attributes are of importance for the users and developers of the software platform. Several important aspects related to different quality attributes are presented.

When using software platforms it is important to understand why the system is designed in a certain way (i.e. the design rationale of the platform). This knowledge can be used to improve the change impact analysis when building variations of products and when updating the platform itself.

Management processes are needed to support development efforts in different phases of platform projects and across the products built on the platform. A way to find improvements is to benchmark the platform process used in the organisation with practices in similar organisations. Another way is to understand how the processes affect the quality by measuring and tracking different quality attributes.

The results from this thesis provide support for quality improvements in software platform development by using the presented approaches.

Acknowledgements

This work was funded by Ericsson Mobile Platforms AB, Sweden (formerly part of Ericsson Mobile Communications AB, Sweden) and by the Swedish Agency for Innovation Systems (VINNOVA), under a grant for the Center of Applied Software Research at Lund University (LUCAS).

I am very grateful to Professor Claes Wohlin for his knowledge and guidance when introducing me to the area of software engineering and research methodology. I appreciate the belief he had in me when enrolling me as a student and providing the opportunity for me to pursue studies to a licentiate degree. I am also very grateful to Dr. Martin Höst for his mentoring and friendship. His sincere engagement and great effort have been crucial factors that brought this thesis to an end. I extend a special thanks to Dr. Lars Bratthall for good collaboration in the early part of the thesis. There had for sure been opportunity for more joint projects if he had not pursued a life in Norway.

I am grateful for the financial support of Ericsson Mobile Platforms AB, which allowed me to spend half of my time as employee at the company as a student.

I will forever be indebted to all my colleagues at Ericsson Mobile Platforms AB and Sony Ericsson Mobile Communications AB (both companies formerly part of Ericsson Mobile Communications AB). They have provided me with data and suggestions that have been invaluable input to the research during these years.

I wish to thank my co-authors for sharing the load and waiting patiently for me to finish my bits and pieces. Also a sincere thanks to all who did the non-glamorous work of reviewing drafts of the thesis and the papers.

A warm thanks to all the current and former colleagues at the Department of Communication Systems for providing a stimulating and fun environment during the work with the thesis, and for just putting up with me in general. The doors in the department have always been exceptionally wide-open, thus creating a good ground for exciting and illuminating discussions whenever I needed encouragement and new impulses. In particular I would thank Professor Ulf Körner, the head of the department, for making this possible.

Finally, I would like to thank my family and friends for their unconditional support and encouragement during this time.

Contents

Introduction

1. Overview

During the last decades, software has become an indispensable part of many commercial products. Products of all kinds, from digital cameras and cellular phones to transaction systems, automobiles and aeroplanes contain more and more software every year. This has meant that software has become a competitive advantage in many industries. Strategic thinking about the functionality, quality and lead-time of the entire product is largely dependent of the methods and tools used in the development and management of the software. Building product platforms is one of the methods that has become a common and crucial element of today's business planning. Product platforms are a set of common components around which a stream of derivative products can be built.

The use of platforms as an engineering concept is not new. It is in many aspects the common practice of the past being rediscovered today by management and developers in various industries. The practice consists of building a line of products around a set of shared components. For example, in the case of the Black & Decker [19] power tools, the shared components have been the motor platform and battery pack platform. Product platforms have also been used with success by the aircraft industry [19], car manufactures [26] and manufactures of consumer electronics [23].

Components that are common for many products and consist of software form a software platform. CelsiusTech Systems AB presented one of the first documented success stories of using software platforms in [2]. To meet a compressed project schedule the engineers built a product line around a software platform.

The term software platform can also be used as an application on which other applications can be executed on, for example an operating system. This definition is neither used nor further discussed in this thesis.

There are risks, which must be managed and assessed when using software platforms. A number of product versions are developed based on the same software platform. The platform must, therefore, be managed and updated according to new requirements if it should be reusable in a series of releases. This means that the platform is constantly changed during its life cycle.

This thesis focuses on identifying and evaluating methods to improve the quality of a software platform in a specific industrial organisation. The intention is that the findings should be general enough to be applied in other organisations with similar software platforms. The software platform quality is in this thesis defined as the ability for a project to use a software platform for different families and versions of software products. The result of the thesis indicates that, although the problem of maintaining and managing the quality in a software platform is a complex task, some basic approaches may be used.

The thesis is organised into two parts. The first part contains an introductory part, which is divided in sections as follows: In Section 2, software platforms are further introduced. Section 3 presents the concept of software quality. The research methodology is summarised in Section 4. In Section 5 the research collaboration between academia and industry is discussed. The research results are presented in Section 6 together with summaries of papers. Finally, Section 7 contains suggestions for future work. The second part of the thesis contains the following papers:

[I] **The Importance of Quality Requirements in Software Platform Development – A Survey**
Enrico Johansson, Martin Höst, Anders Wesslén and Lars Bratthall
Proceedings of *34th Hawaii International Conference on System Sciences* (HICSS-34), pp. 3884-3893, Maui, Hawaii, USA, January, 2001

[II] **Is a Design Rationale Vital when Predicting Change Impact? - A Controlled Experiment on Software Architecture Evolution**
Lars Bratthall, Enrico Johansson and Björn Regnell

Proceedings of *2nd International Conference on Product Focused Software Process Improvement* (PROFES 2000), pp. 126-139, Oulo, Finland, June, 2000

[III] **Benchmarking of Processes for Managing Product Platforms - a Case Study**
Martin Höst, Enrico Johansson, Adam Norén and Lars Bratthall
Proceedings of *Empirical Assessment in Software Engineering* (EASE 2002), Keele, England, April, 2002

[IV] **Tracking Platform Degradation via a Graph Impurity Measure**
Enrico Johansson and Martin Höst
Submitted to the *14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Ischia, Italy, July, 2002

The papers included in this thesis have had small changes and minor improvements, compared to the published versions. For example, text formatting has been changed to provide a common layout of all material.

The following report is related but not included in the thesis:

[V] **Benchmarking at EMP**
Enrico Johansson
LD/EMP/GTX/KX 02:006, Technical report describing the architecture development process at Ericsson Mobile Platform. The report was used in the research done in paper III, Ericsson Mobile Platforms, Lund, Sweden, 2002

2. Software platforms

Many products with high requirements on time to market are developed in a series of different versions, which are part of the same product family. That is, one product family can contain versions of the product, which differ in functionality and complexity, see Figure 1.

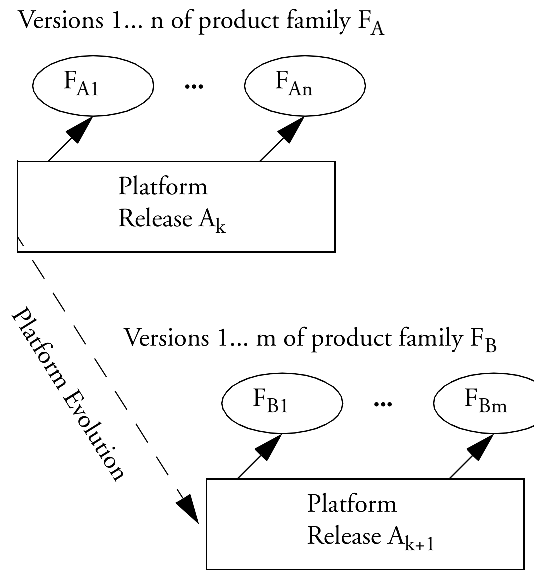
Versions 1... n of product family F_A
F_{A1}
...
F_{An}
Platform Release A_k
Platform Evolution
Versions 1... m of product family F_B
F_{B1}
...
F_{Bm}
Platform Release A_{k+1}

Figure 1. *An illustration of two product families (F_A and F_B) that are based on two consecutive releases of a software platform.*

The advantage of this is that a large portion of the product can be reused in one product family [20] [25]. To achieve an even higher degree of reuse and shortening the time between releases of consecutive product families, different product families could share a common software platform.

The intention is that reuse, within product families and between the different releases of product families, should decrease the development time and the cost of new products. The intention is also that it improves the reliability since already tested and evaluated parts can be used.

However, reuse is difficult to achieve if it is not managed and planned for in earlier products. Software developing organisations need processes that support reuse between different products and between different versions of the same product. This type of processes is often included in a product line architecture [2] [3] [7] [13], which is the basis for a number of versions of a product. A product line architecture is a common architecture for related products or systems developed by an organisation.

With this type of processes, development of a series of products can be summarised as follows. First a software platform is developed. The architecture of the platform should be general enough to be useful in the releases where it should be used. When the software platform has been developed, a number of product projects can be launched. Every product project results in a product version that may be sold on the market. Normally, some product projects are run in parallel, in order to obtain an appropriate pace of new product releases. Each of the product projects uses the software platform as a basis and adds a set of functions that is specific for the release of that version. This type of work is often organised and performed in a number of teams. A team manages the development of the software platform, and the product projects are run by a number of project teams. The objective of the project teams is to develop a new release with a given release date to a given cost. The objective of the platform teams is to manage the software platform and to support the different product projects with functionality packaged in the platform. This means that each of the product projects has a number of requirements on the platform and the objective of the platform team is to provide a platform that meets these requirements.

Ideally the requirements of all product projects on the software platform coincide and the platform team can develop a platform that can be used directly by all development teams. However, for a number of reasons the platform must be enhanced and changed during its lifetime. New requirements are constantly identified and functions implemented in one product project should often be part of future products. This means that the software platform should be constantly enhanced and changed based on the requirements from the product teams. To manage constant changes a number of design rules and design constraints may be formulated in order to limit an eventual loss in quality, in the software platform, caused by the changes. Such loss of quality would make the reuse of the product line between two product families and within the product family very difficult to achieve.

2.1 Architecture

A software platform can be modelled by using a number of different architecture elements and by using a number of different views. In the thesis the software platform has mainly been modelled by using four different elements, see Figure 2. The four elements are components, connectors, interfaces and design rules. Each of the elements is defined as follows:

- Components are elements that contain a selection of functions. A software platform can contain one or several components.
- Interfaces are elements that enable intra-component communication.
- Connectors are the mechanisms used for the communication between the components.
- Design rules are elements that describe and constrain the usage of components, interfaces and connectors.

In order to allow the model to be useful when mapping the objects to arbitrary software platform architectures rather generic elements have been chosen. In addition, the elements chosen are well agreed upon in the architecture community [2] [10] [16].

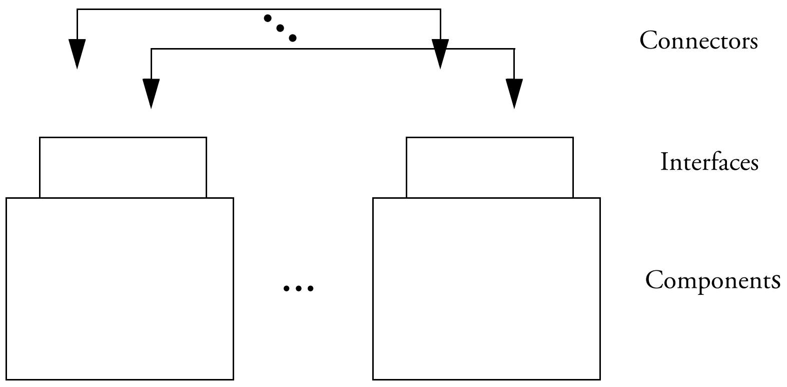

Figure 2. *The interactions between components, interfaces and connectors*

A static deployment view [11] [16] with a model that consists of these four different elements has been used in all papers in the thesis.

2.2 Related work

In the recent years, the area of software product lines and software platforms has been given a lot of attention. This section provides a short summary of two major contributions in this area. The summary is focused on contributions that are related to quality improvements.

In [3] Bosch suggests the use of architecture transformations as a means of improving the quality of a software product line. Four different types of transformation are introduced (imposing an architecture style, imposing an architectural pattern, applying a design pattern and converting quality requirements to functionality). Normally it is necessary to use a combination of these four different types to obtain the required quality improvement. The transformations can be further specialised to improve the quality of software product lines. The specialisation is done in order to achieve three crucial aspects of a product line, (variability, optionality and conflict resolution). The variability aspect is needed when the architectural solution does not satisfy all the requirements members of the product line. An optionality aspect is needed to enable the inclusion of optional functionality in the products. Finally, the third aspect concerns the conflicts that can arise in a software product line. Bosch gives also different examples of techniques improving the run-time quality attributes (i.e. performance and reliability) of software product lines. The exemplified techniques are concerned with caches, memory management, indirect calls and wrappers.

The software platform is by Northrop and Clements [7] defined as the assets that form the basis of the software product line. The development of software platforms is together with product development and management identified as essential activities in the successful scoping of a software product line. The product line development depends on three outputs described as the product line scope, the production plan and the software platform. Variability in product lines is brought up as an essential quality enabler for the software platforms. Mechanisms as a service component framework and a requirement definition hierarchy are proposed as a mean of achieving the variability. Product line management deals with the resource planning, co-ordination and supervision of activities. The management must be committed to both the technical and organisational levels of a software product line. Techniques for collecting metrics and tracking data are also discussed as means to manage the product line process. Northrop and Clements include in [7] part of the extensive work done by the Product Line Initiative at Software Engineering Institute (SEI PLP).

3. Software platform quality

Quality is a software platform and development process prerequisite, which must be in place and continuously improved if successful software product line development is to be achieved. However, the intangible facet of value of quality makes the improvement of it a challenging task. To deal with this, different quality models and quality improvement techniques have been developed. When these models and techniques are used in the software domain, it must be considered that software development is mostly a development skill and not a manufacturing skill. General quality improvement techniques and models used in the manufacturing industry cannot be used directly in the software development.

The discipline of software quality engineering has put focus on both producing new models and modifying traditional models for the software industry. An established software quality model is for example the McCall quality model [18]. The model identifies a number of quality factors from three different ways (operation, revision, transition) of using a software product. Table 1 illustrates the model.

Table 1. McCall quality factors

Product operation	**Product revision**	**Product transition**
Usability Integrity Efficiency Correctness Reliability	Maintainability Testability Flexibility	Reusability Portability Interoperability

There exist several established techniques for assuring and improving the quality of software development. A list of such techniques, without pretending to be complete, can be summarised as follows:

- Testing
- Inspections and reviews
- Software Process Assessment
- Measurements

- Benchmarking
- Project Tracking

Software platforms are used when striving to maintain a number of quality factors in a series of products. Some of these quality factors are concerned with the ability of software to function as a software platform. Other factors are concerned with quality requirements from the domain of the products that are based on the platform (e.g. real-time, embedded, safety critical, enterprise, etc.).

When a company releases new product families it is vital to meet a desired market-window with a quality of the product that matches the expectations of the market. The usage of product line architectures and software platforms is enforced by the potential decrease in lead-time that can be gained. This leads to increasing the possibility of a successful release of a product during the desired market-window. Missing to release a product during a desired market-window or releasing a product family with quality attributes not matching the requirements from the market, can cause major financial losses and even to loose the complete market opportunity for future products.

Since it is believed that software platforms decrease lead-time and increase quality, they are used by an increasing number of companies to meet tight market driven time-plans and quality plans for releases of product families.

Neglecting to keep track of whether the platform is suitable for being used for another release of another product family can lead to an incomplete management of risks. Inappropriate usage of software platforms can lead to a decrease in quality and an increase in lead-time for new product families developed. Therefore, there is a clear need of knowing if the software platform has high enough quality to be the base of a product family.

Software platforms, as all other software artefacts, are subject to quality changes during their life cycle. Considering the high-level nature of the development decision related to replacing and updating elements in the platform, these decisions should preferably be taken from an architectural abstraction level. This is further motivated by the fact that a software platform is the core asset of a product line, and as such incorporates architectural constructions to cater for maintainability, reusability, performance and other quality attributes.

4. Research methodology

Research can be defined as the method of investigation that, if results are obtained and if correctly undertaken, builds knowledge. The investigation done represents an objective investigation of facts about a certain subject.

The research can be performed as either applied research or basic research. Whereas basic or pure research attempts to expand the limits of knowledge, applied research attempts to find the solution to a specific problem. Research can also be classified in many other ways. For example, a distinction between quantitative and qualitative research can be made.

The research done in the thesis, which can be classified as applied research, has been performed to improve the development of software platforms.

The focus in this section is on what methods was used to explore and test the research questions. In the end of the section, the validity and industrial application of the thesis are discussed.

The research questions pursued in the thesis are the following:

- What quality factors are important for users of software platforms?
- What quality factors are important for developers of software platforms?
- How should the software platform design documents be written?
- How should the interactions between software platform stakeholders be optimised?
- How should the platform development and management process be improved?
- How should the software platform degradation be tracked?

4.1 Empirical validation

To every research study, there are a number of threats to the validity. In this section the validity is discussed based on an often-used list of threats to validity [5]. The validity of empirical studies is often evaluated based on four aspects: conclusion validity, internal validity, construct validity, and external validity. These aspects are elaborated below:

- The conclusion validity concerns whether it is possible to draw statistically significant conclusions from the study.
- The internal validity concerns the possibility that the effect (outcome) has been caused by factors unknown to the researchers.
- The construct validity concerns whether the measurements and interviews represent the constructs of interest in the study.
- The external validity of a study concerns the ability to generalise the findings.

In the following paragraphs, each validity concerns is discussed along with a number of considerations that can be seen as threats to the validity of the result presented in the thesis.

Conclusion validity. In this thesis, the number of people that has been involved in the studies is the most serious threat to the conclusion validity. In addition, a related threat concerns the uncertainty and the dispersion of the participants. It will generally be hard to draw valid conclusions from the study if the participants answer differently and the uncertainty is large. If there are few people, dispersion in the answers is hard to interpret. When designing the research the goal has been to minimise these threats by using as many participants as possible.

Internal validity. None of the results in this thesis has been validated by performing a replication in the same environment and with the same instrumentation. Therefore, it cannot be ruled out that the outcome would have been different if the studies had been performed at another point of time. However, great care has been put on the instrumentation to minimise this threat. Great care has also been taken when selecting participants for the studies. For example, when groups were compared, attention was taken to ensure that the groups contained equal level of expertise. Another aspect when selecting participants is that they have always been recruited on voluntary basis. Knowing or controlling all the factors that affect the research is a truly challenging task. It must however be done to maximise the internal validity of the result. The precautions taken are believed to have had a positive effect in achieving greater internal validity.

External validity. A related question concerns the reliability with respect to the environments in the other companies. Although some of the studies were carried out in another company, it is of course, not obvious that the factors affecting the result would be the same. This is due to a different environment. However, we have not identified any factors that indicate that the environment should be principally different from the other companies. To gain better external validity further studies and replications in the area could be carried out.

Construct validity. A threat to the construct validity is that the persons that are interviewed are restricted in their answers, which means that the result actually concerns different constructs than the researchers intend to investigate. In this thesis, we believe that the participating people have been speaking openly. They are guaranteed anonymity and the questions are to the researchers' knowledge in no way sensitive. This, in combination with that the atmosphere at the company is open, means that we do not believe that this kind of threat to the construct validity is large in this thesis.

4.2 Empirical methods used

Experiments, surveys and case studies have been used to validate the result of the research. The methods are by Robson [22] summarised as:

- *Experiments: measuring the effects of manipulating one variable on another variable.*
  A typical feature is that samples of individuals from known populations are studied. The samples are then allocated to different experimental conditions. The conditions are then altered by making changes to one or more variables. Normally measurements can be performed on only small number of variables, when at the same time issuing control on a larger amount of other variables. The analysis of an experiment usually involves hypothesis testing.

- *Survey: collection of information in standardised from groups of people.*
  A typical feature is that samples of individuals from known populations are studied. A relatively small amount of data in standardised form is collected from each individual. A survey usually employs questionnaires or structured interviews to collect the data.

- *Case studies: development of detailed, intensive knowledge about a single "case", or a small number of related cases.*
  A typical feature is to investigate a selection of a single case (or a small number of related cases) of a situation. The case can concern

an individual or group. The study of the case is done in its context. A range of data collection techniques including observations, interview and documentary analysis can be used.

Each of the methods used has its advantages and disadvantages. To decide what method to use is an important step in research design. The choice done will govern the validity expected of the result, and the possibility of the conclusion that can be drawn from the result. Therefore, the method chosen must be based on the research questions that are explored. Table 2 summarises the choices made in the thesis.

Table 2. Summary of research questions and empirical methods

Research question	Empirical method
What quality factors are important for users of software platforms?	Survey
What quality factors are important for developers of software platforms?	Survey
How should the software platform design documents be written?	Experiment
How should the interactions between software platform stakeholders be optimised?	Survey
How should the platform development and management process be improved?	Case study
How should the software platform degradation be tracked?	Case study

4.3 Industrial application

To be able to impact an industrial software organisation with the gained result the research must have high industry validity. Robson [22] classifies this validity from a real world perspective as:

- *Level A: The traditional approach:"Science only".*
  The research is of a theoretical nature. Although the focus can be on solving practical problem, the application of the research is not seen as important and is often left to others to study.
- *Level B: Building bridges between researcher and user.*
  The researcher believes that the eventual outcome of the research has practical implications, and wants to influence the client with

the outcome. The work may be conducted in collaboration with the client and may include giving the client status reports of the ongoing work.

- *Level C: Research-client equality.*
  The researcher and client discuss the problem areas to investigate in collaboration. The client, the researcher or the client and researcher in collaboration may identify the research problem. The work must be performed together with the client, where the client can issues some control over the work done.
- *Level D: Client-professional exploration.*
  A client requests help from a researcher. The collection of data is minimal and the advice or recommendation is based on the researcher's expertise in the area.
- *Level E: Client dominated quest.*
  The client requests help from a specialist or colleague with relevant expertise. The advice or recommendation is given based on current practices or knowledge.

The research questions in this thesis have been investigated with level of application equal to C.

5. Interaction between industry and academia

The interaction between industry and academia has been going on from the very start of the software engineering discipline. There are several examples of industry involvement in academic research projects that have been fruitful for both parties. A win-win situation for both parties is normally the goal for the collaboration between industry and academia. This is however not a situation that happens per se. On the contrary, in order to make it work there are a number of issues that must be taken into consideration. There are a variety of ways to organise the interaction between industry and academia. Different forms of interaction can for example be:

- Workshops and seminars through university-industry research centres.

- Networks to focus on specific areas and needs of the local industry.
- Industry-sponsored academics (Professors, Ph.D. students, etc.)
- Research projects with industrial collaborators

The workshops can be sponsored by industry funding and organised by academia researchers. The objective of the workshops is to encourage attendees to brief each other on recent developments in academia and in industry. Workshops can encourage collaborations between researchers and industry to influence the direction of research projects.

Networks are a way of bringing together participants from industry and academia that share a common area of interest in a common organisation. The overall goal of the networks is to strengthen the way technology is developed and exploited within a regional area by stimulating technology transfer between researchers and industry.

The industry-sponsored academics represent an area of the collaboration where individual faculty positions are funded by industry. Industry can choose to finance the complete cost of a full-time position or it may choose to finance only a part of the cost. Which of the choices is taken affects the level of influence that industry can have on the research area investigated. When a complete research area is of interest the funding may result in the establishment of a professor's chair. When the collaboration is in areas where academia have the expertise and industry need answers to a specialised area or research question it may be most appropriate to sponsor a Ph.D. student. The reason for industry not investigating the problem can be the lack of equipment and expertise in the area. A research contract can be laid out to describe what is expected from the university and what shall be provided by industry.

Research collaborators from industry can be involved in both academic research and education. For example, the aim could be to have courses taught by industry experts. These courses could preferably be in the form of projects; in order to put the emphasis on the application of the theories taught.

In Section 5.1, incentives for collaboration are presented. Finally, Section 5.2 discusses practical considerations when conducting research in industry.

5.1 Incentives for a collaboration

In order for any of these collaborations to be a perceived as a win-win situation, both parties must have realistic incentives when entering them. Incentives are the motives for collaboration, which are based on the beliefs of benefits to be gained. The incentives induce people and organisations to behave in a certain way. Incentives for the collaboration between industry and academia can be classified in the following categories [17]:

- Infrastructure incentives such as technology transfer and professional contacts.
- Economic incentives as for example the trend in the recent years that the Swedish government matches the economical commitment from industry with an equal value.
- Honorific incentives, such as official awards and unofficial public recognition are important driving forces.
- Knowledge incentives, such as training, participation in seminars and workshops and participating in projects.

One interesting issue for collaboration is to understand if the incentives for collaborations were realistic. Realistic incentives are defined as incentives that are based on expectations that came true during the collaboration.

From the listed categories, three incentives are normally mentioned when discussing the reason for academia to collaborate with industry. There is the incentive for academia and in particular for faculty members to gain practical knowledge to improve their own pedagogical function (knowledge incentive). Another incentive is to support and advance their personal research agenda (honorific and economic incentives). Finally, there is the incentive of entrepreneurship that means to capitalise on its own research and intelligence property (infrastructure incentive). All these incentives can be decomposed into the following list of reasons for academia to initiate industry collaborations [17]:

1. Finding supplements funds.
2. Testing the practical applications of the research and theory.
3. Gaining practical insights related to the research done.

4. Strengthening the university's outreach mission.
5. Looking for business opportunities.
6. Finding scenarios that are useful for teaching.
7. Creating student internship and job placement opportunities.
8. Securing funding for research assistants and lab equipment.

When discussing industry incentives the most obvious ones are the knowledge and infrastructure incentives. It is a natural step for industry to take contact with an academic research group when needing answers to specific problems. The incentives oriented towards infrastructure are related to support long-term collaboration goals. Reasons for industry to initiate collaboration with academia can be decomposed into the following list [17]:

1. Solving specific problems.
2. Developing new products and processes.
3. Filing new patents.
4. Improving quality of products.
5. Reorienting the R&D program.
6. Accessing new research (via seminars and workshops).
7. Maintaining an ongoing relationship and network with the university.
8. Searching for new technology.
9. Conducting basic research with no specific applications in mind.
10. Recruiting university graduates.

5.2 Practical considerations

In this section practical consideration of time, management, feedback and intelligence property in research collaborations are discussed. The discussion focuses on performing research in the role as an industrial Ph.D. student. None of the considerations is specifically related to a single research project or case study performed during this work. Instead, it is a summary of guidelines with general applicability.

- *Time:*
  In a software company with projects with tight deadlines, time is always a scarce resource. There might be numerous meetings and other project duties that must be completed during a day at work. An environment like that is far from optimal from a collaborative point of view. Unfortunately, the described environment is common among software companies. There are, however, some practical consideration that can be used to make research collaboration smoother to perform. One practical consideration could be to plan the research session to be performed during a period of the project that is relatively calm. A different aspect of time is the duration of the research session. The longer the research session is intended to be the more in advance should it be planned. Besides working on research projects alone, industrial Ph.D. students have the opportunity to work on short-term industrial projects to gain additional experience. This allows the student to keep in touch with industry practices. Although the intention is for the best, there are difficulties along the way. The main difficulties derive from the different perspectives of projects. Academia stands for the long-term project perspective and industry for the short-term perspective.

- *Management:*
  When starting a collaboration that involves employees in a company, the consent and support from project managers is required. The consent is required even if the company already funds the research. In addition, in discussion with management it is important to highlight the objective of the research done in terms of solving specific problems and values for the company as a whole. Another potential management issue that must be clarified prior to the collaboration is how the researchers' time is managed and prioritised.

- *Feedback:*
  The companies' commitments to a research project are often made by the management. It is on the other hand not the managers but the software engineers that participate in the collaboration. Furthermore, the software engineers are the ones that allow part of their

time into the research collaboration. Therefore, it is important to realise that the engineers must feel that participating in the research gives something of value to them. When the research collaboration solves a specific problem, the answer to the problem is the self-evident feedback. The research collaboration can on the other hand be of a more general aspect and looking at more long-terms aspects. Even if this is the case, there is still a number of ways to give feedback to the participants. For example, training sessions can be included in the research. The training can be done before, during or after the research. A seminar and explanation of the result is the least that should be offered to the participants in the research. If the research is conducted during a long period, it may be a wise approach to give the feedback as intermediate seminars or training sessions.

- *Intellectual Property:*
  Academia and industry share a common goal to produce intellectual property. There are, however, differences in the objective of wanting to achieve the goal and of using the intellectual property. The objective in academia is to use the intellectual property as a tool to advance and disseminate knowledge. The objective in industry is to capitalise the intellectual property in patents and products. That is, the industry has a protective relation to its intellectual property, while academia has a complete opposite relation. Publications of intellectual properties are in academia encouraged and mandatory.

Although considerations have been given to practical problems, it can be that the research collaboration does not pay off. The reason could be that wrong participants have been chosen for the research collaboration. Another reason could be that the company research strategies do not endorse collaboration with academia.

There is a possibility to gain mutual benefits in an academia-industry co-operation if and only if the threat of having the wrong scope and incentives in mind when starting the collaboration are taken seriously, and dealt with appropriately. These threats were thoroughly discussed in this section. Difficulties from practical aspects of the collaboration in the view of time, management, feedback and intellectual properties have been discussed and some guidelines were given.

6. Research result

6.1 Summary of results

The result of the thesis indicates the possibility of quality improvement in several phases and areas of software platform development. The main findings can be summarised as answers to the initial research questions:

- *What quality factors are important for users of software platforms?*
  What quality factors are important for developers of software platforms?
  From studying how different stakeholders prioritise different quality factors, reliability seems to be most important for both the developers as the users of the platform.
- *How should the software platform design documents be written?*
  It is likely that having access to a design rationale expressed in the suggested way in the design documents has a positive impact on both lead-time as well as quality. This fact was experienced by designers when faced with the task of predicting where changes must be performed on an unknown real-time system. In projects where lead-time is important, it is possible that a sharp schedule prohibits the creation of models during design. In that case, writing a design rationale in the suggested manner after initial system release may be a cost effective way to facilitate future system evolution, without prolonging the time to initial system release. This result can easily be incorporated in standard development processes.
- *How should the interactions between software platform stakeholders be optimised?*
  A few rules of thumb for the development of the architecture of software platforms are devised. One stakeholder, such as an architecture group, should be given the uttermost responsibility for the fulfilment of the reliability of the platform. A software platform should in general not be developed as a part of another project, or it is likely that either the software platform becomes unreliable due to lead-time constraints, or the project may become late. Even experienced individuals have varying opinions regarding what is important to have in a software platform. Therefore, techniques that

allow for both the precise communication of what is needed as well as techniques for eliciting these requirements from different stakeholders are needed.

- *How should the platform development and management process be improved?*
  An approach to benchmark two software platform organisations is proposed and proven feasible in a case study. The benchmarking approach consists of two parts. One part is a questionnaire with eight questions customised to elicit improvements. The other parts consist of letting organisations review descriptions of each other's answers to the questionnaire. The majority of the participants acknowledged the questionnaire as positive. The result indicates also that it is possible to use the questionnaire and the cross-wise reviews as a benchmarking approach.

- *How should the software platform degradation be tracked?*
  A measure for product line degradation is proposed and it is shown that the measure is based on sound theoretical properties. In a case study the measure is evaluated and there are indications that the measure can be usable to track product line degradation based on measurements on the software platform. By knowing the gradient of the graph depicted by the proposed measure the degradation of the complete product line can be tracked and predicted.

6.2 Summary of papers

This section lists the abstracts of the individual papers in this thesis.

Paper 1: The Importance of Quality Requirements in Software Platform Development - A Survey

Enrico Johansson, Martin Höst, Anders Wesslén and Lars Bratthall

Proceedings of *34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, USA, January, 2001

This paper presents a survey where some quality requirements that commonly affect software architecture have been prioritised with respect to cost and lead-time impact when developing software platforms and when using them. Software platforms are the basis for a product-line, i.e. a collection of functionality that a number of products is based on. The survey has been carried out in two large software developing organisations using 34 senior participants. The prioritisation was carried out using the Incomplete Pairwise Comparison method (IPC). The analysis shows that there are large differences between the importance of the quality requirements studied. The differences between the views of different stakeholders are also analysed and it is found to be less than the difference between the quality requirements. Yet this is identified as a potential source of negative impact on product development cost and lead-time, and rules of thumb for reducing the impact are given.

PAPER 2: Is a Design Rationale Vital when Predicting Change Impact? - A Controlled Experiment on Software Architecture Evolution

Lars Bratthall, Enrico Johansson and Björn Regnell

Proceedings of *Second International Conference on Product Focused Software Process Improvement*, Oulo, Finland, June 2000

Software process improvement efforts often seek to shorten development lead-time. A potential means is to facilitate architectural changes by providing a design rationale, i.e. a documentation of why the architecture is built as it is. The hypothesis is that changes will be faster and more correct if such information is available during change impact analysis. This paper presents a controlled experiment where the value of having access to a retrospective design rationale is evaluated both quantitatively and qualitatively. Realistic change tasks are applied by 17 subjects from both industry and academia on two complex systems from the domain of embedded real-time systems. The results from the quantitative analysis show that, for one of the systems, there is a significant improvement in correctness and speed when subjects have access to a design rationale document. In the qualitative analysis, design rationale was considered helpful for speeding up changes and improving correctness. For the other system the results were inconclusive, and further studies are recommended in order to increase the understanding of the role of a design rationale in architectural evolution of software systems.

PAPER 3: Benchmarking of Processes for Managing Product Platforms - a Case Study

Martin Höst, Enrico Johansson, Adam Norén and Lars Bratthall

Proceedings of *Empirical Assessment in Software Engineering*, (EASE 2002), Keele, England, April, 2002.

This paper presents a case study where two organisations participate in a benchmarking initiative in order to find improvement suggestions for their processes for managing product platforms. The initiative is based on an instrument which consists of a list of questions. It has been developed as part of this study and it contains eight major categories of questions that guide the participating organisations to describe their processes. The descriptions are then reviewed by the organisations cross-wise in order to identify areas for improvement. The major objective of the case study is to evaluate the benchmarking procedure and instrument in practice. The result is that the benchmarking procedure with the benchmarking instrument was well received in the study. We can therefore conclude that the approach probably is applicable for other similar organisations as well.

PAPER 4: Tracking Degradation in Software Product Lines through Measurements of Design Rule Violations

Enrico Johansson and Martin Höst

Submitted to the *14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Ischia, Italy, July, 2002

In order to increase reuse, a number of product versions may be developed based on the same software platform. However, the platform must be managed and updated according to new requirements if it should be reusable in a series of releases. This means that the platform is constantly changed during its life cycle, and changes can result in degradation of the platform. In this paper, a measurement approach is proposed as a means of tracking the degradation of a software platform and consequently in the product line. The tracking approach is evaluated in an industrial case study where it is applied to a series of different releases of a product. The result of the case study indicates that the presented approach is feasible.

7. Future work

Examples for further work in these areas are given for benchmarking the platform process, tracking degradation in product lines and performance investigations in software platforms.

Further work can be done in benchmarking the platform process. For example, the real improvement effect of the benchmarking can be studied and the use of the benchmarking instrument may be extended. It would be interesting to investigate if the benchmarking instrument can be used for self-assessment or if a third party could perform the assessment. The following research questions might be pursued:

- Can a benchmarking approach be used for self-assessment of the development process for software platforms?
- Does a benchmarking approach have significant measurable improvement on the development process for software platforms?

Further work can be done in the area of tracking product line degradation. A significant challenge for such a tracking is to depict the degradation quantitatively and relate the quantitative values to actions performed in the platform development process. The approach can also be investigated in order to track the degradation of specific quality attributes (e.g. degradation of maintainability, degradation of performance, degradation of usability, etc.). The following research questions might be pursued:

- Can a software platform measure be used as a general approach to track the degradation of software product lines?
- How can the gradient of a degradation measure be related to the platform development process?
- Can a software platform measure be used to track the degradation of specific quality attributes (e.g. performance, reliability, usability, etc.) in software product lines?

Further work can be done in optimising the performance of the platform without sacrificing the other platform specific aspects. Performance is an important quality attribute of a software platform. This is, in particular, true for an embedded system as for example a software platform for hand-held communication devices. Also when constructing the software platform to primarily deal with other quality attributes than performance, for example maintainability and reusability, there is a strong cost and real-time incentive to avoid performance penalties. Therefore, there is a need to predict the performance of a software platform, or at least a need to compare available architectures of platforms for best performance characteristics. The research may deal with early performance assessments and prediction of software architectures in order to explore the described problem areas. The following research questions might be pursued:

- How should an evaluation method for investigating the performance of an architecture implementation be designed?
- How should a method that is used to compare different architectures' performance look like?
- What performance characteristics can be estimated by evaluating an architecture design?
- What guidelines for architecture design can be proposed by investigating the performance of an architecture implementation?

The research field of quality improvement in software platform development is still a wide-open arena. Many quality factors and their implications in various phases of the development process need to be explored and given research attention to.

8. References

[1] Basili, V.R., "The Experimental Paradigm in Software Engineering", In Experimental Software Engineering Issues: Critical Assessment and Future Directives, Edited by Rombach, D., Basili, V.R., Selby, R.W., Lecture Notes in Computer Science, pp. 3-12, August, 1993

[2] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practise*. Addison Wesley, 1998

[3] Bosch, J., *Design & Use of Software Architectures: Adopting and Evolving a Product-line Approach*, ACM Press/Addison Wesley, 2000

[4] Briand, L.C., Morasca, S., Basili, V.R., "Measuring and Assessing Maintainability at The End of High Level Design", *Proceedings of International Conference on Software Maintenance (ICSM'93)*, pp. 88-97, 1993

[5] Campbell, D., Stanley, J., *Experimental and Quasi-Experimental Designs for Research*, Chicago, IL: Rand-McNally, 1963

[6] Carriere, S.J., Kazman, R., Woods, S., "Assessing and Maintaining Architectural Quality", *Proceedings of Third European Conference on Maintenance and Reengineering (CSMR'99)*, pp. 23-30, Amsterdam, 1999

[7] Clements, P., Northrop, L.M., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001

[8] Gall, H., Hajek, K., Jazayeri, M., Detection of Logical Coupling based on Product Release History, *Proceedings of International Conference on Software Maintenance (ICSM'98)*, pp. 190-198, Washington D.C, 1998

[9] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Abstraction and Reuse of Object-Oriented Design", *Proceedings of 7th European Conference on Object Oriented Programming (ECOOP'93)*, pp. 406-431, Springer Verlag, 1993

[10] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[11] Hofmeister, C., Nord, R., Soni, D., *Applied Software Architecture*, Addison-Wesley Longman, 2000

[12] IEEE. *IEEE Standard 610.12-1990:* IEEE Standard Glossary of Software Engineering Terminology, 1990

[13] Jazayeri, M., Ran, A., van der Linden, F., *Software Architecture for Product Families: Principles and Practise*, Addison-Wesley, 2000

[14] Kazman, R., "Tool Support for Architecture Analysis and Design". *Proceedings of 2nd International Workshop on Software Architectures*, pp. 94-97, ACM Press, 1996

[15] Krikhaar, R., Postma, A., Sellink, A., Stroucken, M., Verhoef, C., A Two-Phase Process for Software Architecture Improvement, *Proceedings of IEEE International Conference on Software Maintenance (ICSM '99)*, pp. 371 -380, 1999

[16] Kruchten, P.B., "The 4+1 View Model of Architecture", *IEEE Software*, Vol. 12, No. 6, pp. 42-50, November, 1995

[17] Lee, Y.S., "The Sustainability of University-Industry Research Collaboration: An Empirical Assessment, *Journal of Technology Transfer, No. 25*, pp. 111-133, Kluwer Academic Publishers, The Netherlands, 2000

[18] McCall, J.A., Richards, P.K., Walters, G.F., "Factors in Software Quality" *RADC TR-77-369*, 1977, Vols. I, II, III, US Rome Air Development Center Response NTIS AD/A-049 014, 015, 055, 1977

[19] Meyer, M., Lehner, A., *The Power of Product Platforms*, The Free Press, New York, 1997

[20] Parnas, D.L., "On the Design and Development of Program Families", *IEEE Transactions on Software Engineering*, No. 2, March, 1976

[21] Perry, D.E., Wolf, A.L., "Foundations for the Study of Software Architecture". *Software Engineering Notes*, Vol. 17, No. 4, pp. 40-52, October, 1992

[22] Robson, C., *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*, Blackwell Publishers Ltd., United Kingdom. 1993

[23] Sanderson, S., Uzumeri M., "Managing Product Families: The Case of the Sony Walkman," *Research Policy,* No. 24, pp. 761-782, 1995

[24] Shaw, M., Garlan D., *Software Architecture – Perspectives on an Emerging Discipline*, Prentice Hall, USA, 1996

[25] Weiss, D.M., Lai C.T.R., *Software Product-line Engineering: A Family-Based Software Development Process,* Addison-Wesley, Reading, MA, USA, 1999

[26] Womack, J., Ones, D., Rooms, D., *The Machine that Changed the World*, New York: Harper-Colons, 1991

Paper I:

The Importance of Quality Requirements in Software Platform Development – A Survey

Enrico Johansson, Martin Höst, Anders Wesslén and Lars Bratthall

Proceedings of HICSS-34, *Hawaii International Conference on System Sciences*, Maui, Hawaii, USA, January, 2001

Abstract

This paper presents a survey where some quality requirements that commonly affect software architecture have been prioritized with respect to cost and lead-time impact when developing software platforms and when using them. Software platforms are the basis for a product-line, i.e. a collection of functionality that a number of products is based on. The survey has been carried out in two large software developing organizations using 34 senior participants. The prioritization was carried out using the Incomplete Pairwise Comparison method (IPC). The analysis shows that there are large differences between the importance of the quality requirements studied. The differences between the views of different stakeholders are also analysed and it is found to be less than the difference between the quality requirements. Yet this is identified as a potential source of negative impact on product development cost and lead-time, and rules of thumb for reducing the impact are given.

1. Introduction

Time to market for new products is an important business driver for many organizations [5, 12, 23, 24, 25]. Sometimes, the market can be an all-or-almost-nothing market, and there can be a clear benefit from being first to market, i.e. having the first-mover advantage [20]. For example, in [2] an organization stated that being only one week late to market could make a product introduction fail completely, and another organization stated that being just three months ahead of its closest competitor allowed it to become world market leader.

A commonly used technique to compress product development lead-time is to develop a software platform, and then make minor changes in order to release a product that is perceived as "new" [27]. Thus a software platform has an impact on the cost, the development lead-time and the overall quality of several generations of products.

A software platform has several qualities, such as its maintainability, its efficiency etc. Since a software platform has an impact on several generations of products, the set of qualities in the platform may be more important than the set of qualities in a single product. At the same time, there is a cost associated with achieving these qualities in a software platform.

In order to find out which qualities are considered the most expensive to obtain, as well as which are the most wanted, a survey has been conducted. The study was conducted in two steps. First a pilot study was carried out in academia (A) in order to improve the instruments later used for data collection. As a second step, two industrial organizations, subsequently denoted Organization B and Organization C, have participated in this study. Organization B develops large transaction systems, with high reliability and security demands. Organization C develops hand-held consumer devices for production in large numbers. Both organizations are considered representative for large organizations that develop and use software platforms, where time to market and reliability are major business drivers.

In each of these organizations, there are several stakeholders involved in determining the qualities needed in the software platform and the products based on the platform. If these stakeholders have different views on what qualities are expensive to create in a software platform, and which have a positive impact when creating the products based on the platform, there is a risk that the software platform is sub-optimized. This can result in longer lead-time and unnecessary costs when developing products based on the software platform. This study investigates three roles at Organization B (architects, system designers, and marketing), and two roles at Organization C (architects and system designers). The purpose is to identify differences between stakeholders. In summary, the research questions of this study can be summarized as:

RQ1: Is there a difference in how various stakeholders prioritize quality requirements when developing a software platform?

RQ2: Is there a difference in how various stakeholders prioritize quality requirements, that they want to be present in a software platform when using it for developing a new product?

In Section 2, the method used to investigate the research questions is explained and in Section 3, results are presented. In Section 4, conclusion and applications of results are presented.

2. Method

The strategy used to answer the research questions is a replicated survey study. The planning of the study is outlined in Section 2.1. The operation of the study is described in detail in Section 2.2. In Section 2.3, the procedures for data analysis and presentation are described. In Section 2.4, threats to the validity of the study are discussed.

2.1 Planning

In order to identify which stakeholders and which quality requirements to study, a group of people with extensive industrial and/or research background elicited a list of stakeholders which were known to have an interest in the development or evolution of systems in the two organizations studied. The stakeholders all have a large impact on the elicitation and prioritization of quality requirements. Considering these stakeholders, a number of quality requirements from [14, 17] were selected for investigation. The criteria for selection was that they were believed by the expert group to both have a large impact on product development lead-time, product development cost and product quality and also that they were within the scope of this study.

The definitions in [14, 17] of the quality requirement studied proved to be hard to understand in the pilot study performed before studying the two industrial organizations. Therefore, using these definitions without any modification would imply taking a considerable risk, since there was a risk that participants would not understand them sufficiently. To secure that a common understanding would be achieved, the definitions from [14, 17] have been slightly rephrased. During this process the questionnaires used for data collection were iteratively tested against practitioners who later did not participate in the study. The quality requirements and the final definitions selected for study are shown in Table 1, and the stakeholders studied are shown in Table 2. Apart from these definitions, lead-time is explicitly defined as “the duration of a project. The starting point of the project is when it has been decided that a development project should take place, and ending point is when a product can be used by an end-user, such as a user of a hand-held consumer device or a user of an Internet bank system.”. Cost is defined as “the amount of person-hours needed to accomplish something”. A software platform is defined as “the first in a product-line, which is a collection of functionality that a number of products is to be based on”.

Table 1. Quality requirements studied. (Based on the ISO-9126 quality model).

Id.	Requirement	Definition
I	Efficiency	The architecture shall solve stated or implied needs efficiently with respect to hardware usage (e.g. memory usage and CPU load).
II	Functionality	The architecture shall satisfy the stated or implied needs.
III	Reliability	The architecture shall perform its required functions correctly.
IV	Usability	The effort to understand and use the architecture correctly shall be low.
V	Reusability	It shall be possible to reuse the architecture in other applications in the same environment.
VI	Maintainability	The effort required to make changes to the architecture shall be low.

Table 2. Stakeholders studied.

Name	Definition
Architect	An architect has a high level design responsibility, and is generally more experienced than a system designer.
System designer	A person who is primarily concerned with programming and fine-grained design.
Marketing	A person who is primarily concerned with marketing of products. This role is only studied in Organization B.

2.2 Operation

Data collection has taken place in several steps. First, a questionnaire was given to all participants. On completion, the questionnaire was sent back to the experimenters. The respondents have been given the possibility to comment orally and in writing on their answers. After filling in the questionnaires, participants have been able to clarify answers and provide more input through non-directive interviews [7] in order to capture unexpected information. In non-directive interviews, there is no pre-specified set of questions, nor is there any schedule.

From previous experience [2], it is known that questionnaires must be very clear, or there is a high risk that they will be erroneously interpreted, thus reducing the validity of findings. For this purpose, a pilot study among seven participants (Ph.D. students/Ph.D.s in software engineering) has taken place. The pilot study resulted in modifications to the questionnaire, that made it more clear to the participants. Especially, the concepts of lead-time, cost and software platform were defined in a clear way.

The questionnaire has two parts. The first part contains simple questions that are used to establish the accountability of the participants. This part also functioned as a primer, i.e. a mind opener. The second part has six forms for comparing the quality requirements in Table 1 on a nine-step scale. The scale was exemplified and explained in the questionnaire, and the respondents were encouraged to use the entire scale. The questions asked are listed in Table 3. The order of questions, as well as the order of comparisons made to answer each question are random for each participant.

2.3 Analysis

The main analysis method applied to the quantitative data collected through the questionnaires is a variant of the Analytic Hierarchical Process (AHP) [15, 22], called the Incomplete Pairwise Comparison Method (IPC) [9, 10]. The AHP was developed to improve decision-making through prioritizing items pair-wise, rather than prioritizing all items at once. Thus the method simplifies complex decision-making.

The AHP method is used to pairwise compare a set of *n* objects. For every pair of objects, a subjective comparison should be made and it should be decided how large the difference between the objects is accord-

Table 3. Aspects studied and questions asked.

Aspect id.	Question. Requirements 1 and 2 refer to those in Table 1.
LTDEV	Which of requirement 1 and requirement 2 do you consider the most important in terms of that it increases the lead-time in the creation of a new software platform?
COSTDEV	Which of requirement 1 and requirement 2 is the most expensive to create in a new software platform?
QUALDEV[a]	Which of requirement 1 and requirement 2 in the software platform do you think is the most important to focus on when creating a new software platform?
LTUSE	Which of requirement 1 and requirement 2 in the software platform decreases the lead-time the most when creating the new software product?
COSTUSE	Which of requirement 1 and requirement 2 in the software platform decreases the cost the most of a new software product based on the software platform?
QUALUSE[a]	Which of requirement 1 and requirement 2 in the software platform do you think is the most important property for your work when developing a new software product based on the software platform?

a. The questions for aspects QUALDEV and QUALUSE are intentionally rather loosely defined. In that way the answer to each question is affected by the complete set of requirements and the business situation of each company, as known to the participants.

ing to a pre-defined scale. The comparisons are coded and placed in a comparison matrix. Based on the comparison results a priority vector can be calculated. The priority vector consists of *n* items, which summarize to 1 and describe the relative importance of the *n* objects that are compared.

According to the original AHP method, every possible pair of objects should be compared, i.e., if there are *n* objects, $n(n-1)/2$ comparisons should be made. However, in [3] it is shown that the result is not negatively affected if up to half of the comparisons are removed randomly, i.e. by using the (IPC) method [9, 10]. The original AHP method is intended to be used by one person or one group of people, who for every comparison decides one answer based on consensus. In [6] it is shown how to instead aggregate the result of individual comparisons after the comparisons have been made.

In this study, comparisons have been made individually by the participants and 6 out of 15 comparisons have been omitted randomly in order to lower the number of comparisons that every person has to make.

In the pilot study mentioned in Section 2.2, no comparisons were omitted. The results from AHP analysis of this data-set has been compared to the results of an IPC analysis of the same data-set with random data removed. In most cases, there is a slight modification of the absolute results computed using AHP and IPC. In most cases, the ordering of average results is the same, while individual results vary more. It is concluded that the ordering between the average weight assigned to the objects studied has a high degree of trustworthiness, if the absolute difference between the elements is less than 0.02. In other cases, the IPC used in this study may show an incorrect result.

The qualitative data collected through comments in the questionnaires and through the interviews has been analytically analysed, and are mainly used to describe the various systems.

A number of tests has been used[1], notably two-tailed Kruskal-Wallis tests [16], Analysis of Variance (ANOVA) [19] and Least Significant Difference tests (LSD) [19]. The Kruskal-Wallis test is non-parametric and does not make any assumptions about the distribution or the variance of the data collected. This test is used to initially get a picture of the data analysed. The ANOVA makes two assumptions: The variance in the groups being compared must be equal, and the data must be distributed according to a normal distribution. If both a Kruskal-Wallis test and an ANOVA on the same data-set show the same result, LSD tests can be trusted as it is assumed that the preconditions for the ANOVA have been met well enough. LSD tests have been used to identify what contributes the most to the results of an ANOVA.

Since this study is explorative rather than hypothesis confirming, a relatively large number of statistical tests are performed on data produced by the same participants. This introduces the problem of multiplicity, i.e. the risk that a test yields a significant result by chance due to the number of tests performed. There are several ways to adjust for this risk. For example, the Bonferroni adjustment [18] is a conservative adjustment that states that if a pre-set significance level at e.g. $p=0.10$ is used, performing ten tests would adjust the significance-level to *0.010*. A less conservative method is Holm's method [13], which suggests ordering the results from tests and adjusting the significance level required by the tests according to the initial level, the number of tests, and the ordering of the results from the tests. In this paper, neither Bonferroni adjustment nor Holm's method is used, but the reader should be aware of the multiplicity problem when interpreting data.

1. SPSS 10.0 has been used for all statistical analysis, with the exception for the AHP analysis.

Boxplots are used to present the data. How to interpret boxplots is shown in Figure 1. Note that the values outside the whisker-range are not treated as outliers in the statistical analysis.

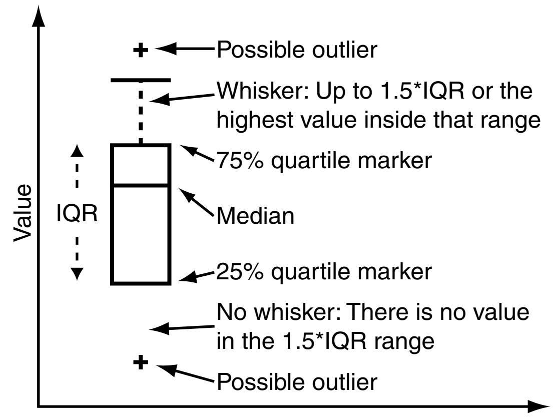
Possible outlier
Whisker: Up to 1.5*IQR or the highest value inside that range
75% quartile marker
Median
25% quartile marker
No whisker: There is no value in the 1.5*IQR range
Possible outlier
IQR
Value

Figure 1. *Interpretation of boxplots.*

2.4 Threats to the validity

The validity of the findings reported depends to a large extent on how well threats have been handled. Four types of validity threats [28] are analysed, namely threats to conclusion validity, construct validity, internal validity and external validity.

Conclusion validity concerns the composition of participants and the statistical analysis. The validity of material gathered through the questionnaire and the interviews is highly dependent on the experience of the respondents. All of the participants have been several years in their organization. Some of the participants are the most senior in their role in their organizations.

Since all participants are anonymous to all but the experimenters, there is reason to believe that participants have answered according to their best knowledge. There should be no social pressure to answer in a particular way.

Interviewer bias cannot be completely removed, but since both questionnaires and interviews have been used, it is believed that the impact of interviewer bias has been minimized. The instrumentation was carefully reviewed with respect to understandability, and a pilot study was carried out as described in Section 2.2. None of the reviewers or the participants in the pilot study are participants in the survey. The reviews and the pilot study are believed to have removed flaws in the questionnaires and the interview guide.

The problem of multiplicity when using several statistical tests on the same data set is addressed in Section 2.3. It was expected that statistical significance would be low. Therefore as experienced participants as possible were included in this study, a decision based on the belief that experienced stakeholders are more secure in their role than inexperienced stakeholders are.

Construct validity concerns whether we measure what we believe we measure. The multiple method approach should reduce the mono-method bias, i.e. relying on data provided by only one data collection method. Care has been taken to provide definitions on important concepts used in the questionnaire. Thus, the threat against the construct validity originating from that participants have different systems in mind and therefore relate differently to the definitions and concepts given should be minimized. It is, however, not possible to completely remove this threat. The randomization questions as well as the individual comparisons within each question should balance any hypothesis guessing. The full anonymity provided should minimize the risk for evaluation apprehension, i.e. attempts to "look better". Also, all participants have been asked how much of the questions in the questionnaire used they believe they understand, and all believe that they have understood the majority of the questions correctly.

Internal validity concerns matters that may affect the measured variables outside the knowledge of the researcher.

The risk for maturation has been explicitly taken into account, by making sure in advance that the questionnaire took not more than 40 minutes to complete. The use of IPC rather than AHP has reduced the number of questions, thus also decreasing the risk for maturation. Selection effects due to how the participants have been selected can never be completely out-ruled, but to the best of our knowledge, the selection process should be adequate. No participant has left the study; i.e. there has been no mortality during the survey. The use of the AHP should reduce the risk for social threats, as should the full anonymity provided. Social threats are threats that may cause a participant to deliberately skew his answers to questions in a particular direction.

External validity concerns generalization of the findings to other contexts and environments than the one studied. Following Robson [21], external validity can be compromised if sampling is not done appropriately, or if mortality is high. Section 1 gives a conservative view on where external validity should be high, i.e. in large organizations where TTM is a major business driver.

Given the amount of thought put into methodology, and considering the high level of experience of the participants, there is reason to believe that the validity is high within a population close to the characteristics described in Section 1. The main threat to validity is probably the use of subjective data rather than objective measurements. Given the large variance in estimations made by some population, shown in for example [11, 26], this is a serious threat. This threat has been addressed through discussing with experienced participants, and by using the IPC.

A valid objection by readers of this survey may be that only a particular set of quality requirements has been assessed, which is a correct observation. However, the choice seems sound given the purpose.

3. Results and analysis

In this section, the results of the survey are presented. Particular results are denoted *Rn*. An overall discussion of findings is presented in Section 3.1. RQ1 is analysed in Section 3.2. In Section 3.3, RQ2 is analysed.

3.1 Overall discussion

For each of the datasets from Organization B and Organization C, an ANOVA has been performed in order to detect differences in how stakeholders prioritize the quality requirements (Table 1), with respect to each aspect (Table 3). No such difference has been detected. Yet, there is a statistically significance difference ($p<0.10$) in the way the stakeholders have graded the importance of the quality requirements for some of the aspects, which is detected both by the ANOVA test and the non-parametric Kruskal-Wallis test. This can be an indication of differences in the way different stakeholders prioritize the quality requirements with respect to various aspects. The data is visualized in Figures 2 and 3 and statistics are summarized in Tables 4 and 5. Naturally this assumption must be regarded with caution, since it is not detectable by the ANOVA test.

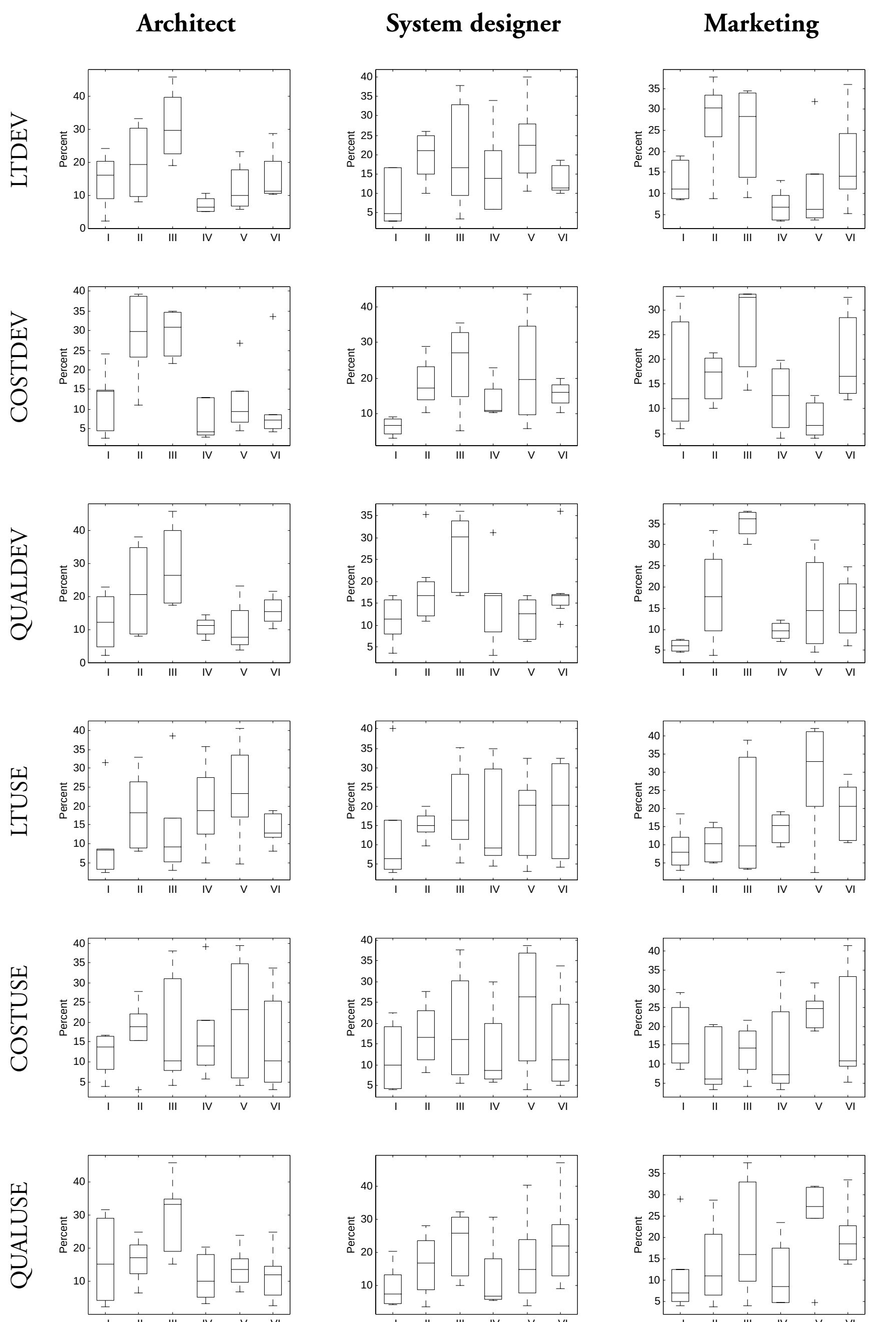

Figure 2. *Organization B, prioritization of quality requirements I to VI with respect to role and aspect.*

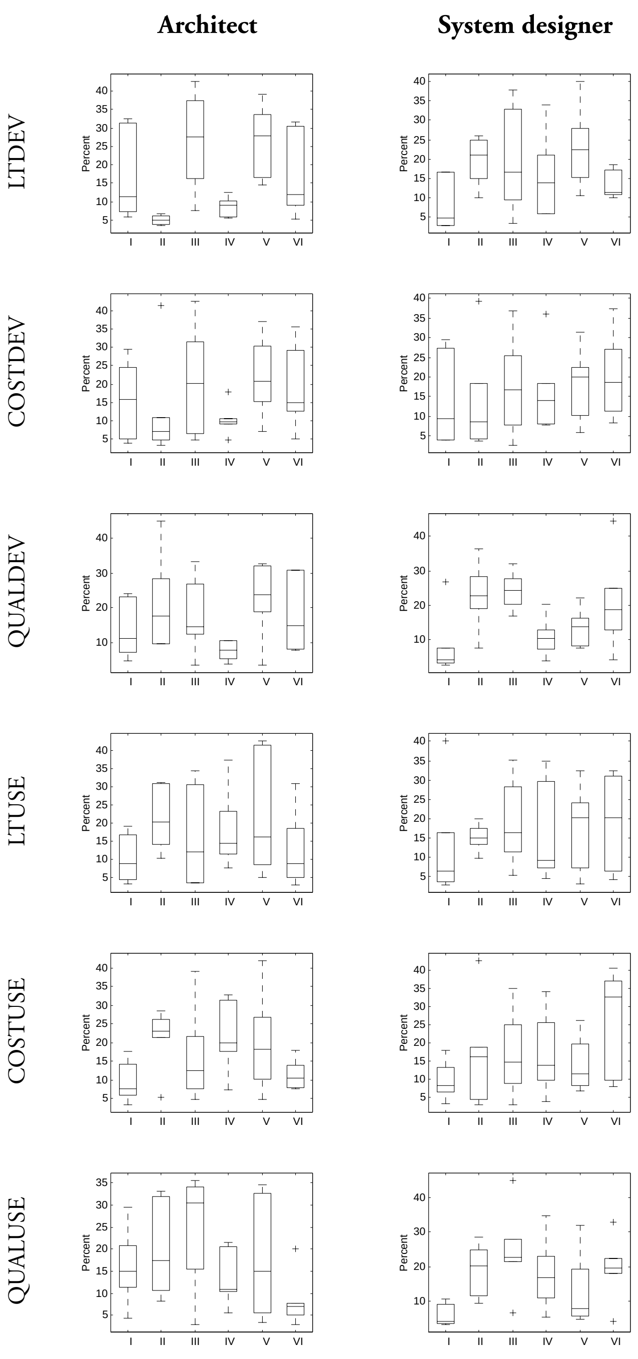

Figure 3. *Organization C, prioritization of quality requirements I to VI with respect to role and aspect.*

Table 4. Results related to the development of a software platform. p-values are given for tests (p<0.10 is highlighted).

		Organization B - Figure 2			Organization C - Figure 3	
		Architect	**System designer**	**Market**	**Architect**	**System designer**
	Participant ID	BA-BH	BI-BO	BP-BW	CA-CF	CG-CL
LTDEV	Kruskal-Wallis test	**0.052**	0.205	**0,028**	**0.001**	0.383
LTDEV	ANOVA test	**0.032**	0.268	**0.026**	**0.001**	0.650
LTDEV	Order of prioritization, highest first	III, II, VI, I, V, IV	V, III, II, IV, VI, I	II, III, VI, I, V, IV	V, III, VI, I, IV, II	II, V, VI, III, I, IV
LTDEV	An LSD indicates a difference between quality requirements priorities	I-III, II-IV, III-IV, III-V, III-VI	Criteria for LSD not fulfilled.	I-II, I-III, II-IV, II-V, III-IV, III-V	I-II, I-III, I-V, II-III, II-V, II-VI, III-IV, III-VI, IV-V, V-VI	Criteria for LSD not fulfilled.
COSTDEV	Kruskal-Wallis test	**0.017**	0.123	0.271	0.328	0.761
COSTDEV	ANOVA test	**0.002**	0.162	0.254	0.436	0.914
COSTDEV	Order of prioritization, highest first	II, III, I, V, VI, IV	III, V, II, VI, IV, I	III, VI, I, II, IV, V	V, III, VI, I, II, IV	VI, V, III, IV, I, II
COSTDEV	An LSD indicates a difference between quality requirements priorities	I-II, I-III, II-IV, II-V, II-VI	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.
QUALDEV	Kruskal-Wallis test	0.184	**0.019**	**0.026**	0.204	**0.012**
QUALDEV	ANOVA test	0.109	**0.004**	**0.019**	0.164	**0.012**
QUALDEV	Order of prioritization, highest first	III, II, VI, I, IV, V	III, VI, V, II, IV, I	III, VI, II, V, IV, I		III, II, VI, V, IV, I
QUALDEV	An LSD indicates a difference between quality requirements priorities	Criteria for LSD not fulfilled	I-II, I-III, I-VI, II-V, III-V, III-VI, V-VI	I-III, I-VI, II-III, III-IV, III-V, IV-VI	Criteria for LSD not fulfilled.	I-II, I-III, I-VI, II-IV, II-V, III-IV, III-V, IV-VI

3.2 RQ1 - Impact of software platform qualities when developing a software platform

This section compares the different views on the studied quality requirement when developing a software platform. The results presented are based on both statistically significant findings and observed trends in the dataset.

R1 In Organization B, marketing considers functionality (quality requirement II) to be the quality requirement of those studied that has the highest impact on lead-time when developing a software platform, closely

Table 5. Results related to the use of a software platform. p-values are given for tests (p<0.10 is highlighted).

		Organization B - Figure 2			**Organization C - Figure 3**	
		Architect	**System designer**	**Market**	**Architect**	**System designer**
LTUSE	Kruskal-Wallis test	0.246	0.781	0.246	0.291	**0.055**
	ANOVA test	0.712	0.951	0.584	0.420	**0.056**
	Order of prioritization, highest first	V, III, IV, II, VI, I	III, VI, V, IV, II, I	V, III, IV, II, VI, I	V, II, IV, III, VI, I	V, IV, II, I, VI, III
	An LSD indicates a difference between quality requirements priorities	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	I-IV, I-V, III-IV, III-V, IV-VI, V-VI
COSTUSE	Kruskal-Wallis test	0.562	0.819	0.308	0.460	0.339
	ANOVA test	0.540	0.774	0.406	0.829	0.210
	Order of prioritization, highest first	V, IV, VI, II I	V, III, II, VI, IV, I	V, VI, I, IV, III, II	IV, II, V, III, VI, I	VI, (II/III), IV, V, I
	An LSD indicates a difference between quality requirements priorities	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.
QUALUSE	Kruskal-Wallis test	0.039	0.293	0.103	0.185	**0.019**
	ANOVA test	0.005	0.379	0.108	0.128	**0.031**
	Order of prioritization, highest first	III, I, II, V, VI, IV	III, VI, V, II, IV, I	V, VI, III, II, IV, I	III, II, V, I, IV, VI	III, VI, II, IV, V, I
	An LSD indicates a difference between quality requirements priorities	I-III, II-III, III-IV, III-V, III-VI	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	Criteria for LSD not fulfilled.	I-II, I-III, I-VI, I-VI, III-V

followed by reliability. Architects, on the other hand, consider that reliability takes the longest lead-time. System designers believe that achieving functionality and reusability takes the longest lead-time.

R2 In Organization B, it can be observed that architects and system designers have a relatively large difference in how they perceive the cost of achieving reusability (quality requirement V), where system designers consider the cost to be higher. This may be due to that the two groups normally work at different architectural aggregation levels [1], where architects work at higher levels. At the same time, marketing believes that achieving reusability is the cheapest quality to achieve of those studied.

R3 In Organization B, it can be observed that all three groups have prioritized reliability as the most important to focus on when developing a software platform. This observation is emphasized by the significant difference in prioritizations among system designers and marketing.

R4 In Organization C, there are similarities (Figure 3) between the prioritization with respect to cost (COSTDEV) and lead-time (LTDEV) aspects for both system designers and architects.

R5 Architects in Organization C prioritize reliability and reusability as the two properties that take the longest lead-time to achieve. The system designers give a much more blurred picture. This is evident in the statistics in Table 4 where no statistically significant difference in the system designers' prioritization with respect to lead-time (LTDEV) can be found.

R6 Regarding what increases cost the most in developing a software platform in Organization C, there is no clear answer for neither the architects nor the system designers. However there is a significant difference in how architects have graded the quality requirements. It seems that architects consider neither functionality nor usability costly to achieve.

R7 There is a significant difference in how system designers in Organization C prioritize quality requirements from how the architects prioritize the quality requirements. They prioritize reliability when developing a software platform. It can be observed that architects in Organization C prioritize reusability instead.

3.3 RQ2 - Impact of software platform qualities when using a software platform

This section compares the impact of the qualities studied, in a software platform used in a new project. The results presented are based on both statistically significant findings and observed trends in the dataset.

R8 In Organization B, all three types of stakeholders consider that the maintainability of the software platform has the largest impact on decreasing the lead-time in a project that builds upon the software platform. The marketing role emphasizes this more clearly than system designers and architects as seen in the boxplots related to lead-time (LTUSE) in Figure 2.

R9 It can be observed that in Organization B, all three stakeholders consider efficiency the least important factor in a software platform with respect to lead-time reduction in a new software product. However, in interviews, it was explained that a "good enough" efficiency in the platform is mandatory for its use, therefore it was not prioritized the highest.

R10 In organisation B, the impact of quality requirements on cost and lead-time in a new software project is considered similar for all three groups. This indicates that there is a close correlation between cost and lead-time in projects where a software platform is used.

R11 Generally in Organization B, architects and designers believe that reliability is the most important quality in a software platform for the success of a new software project. However, marketing prioritizes reusability much higher than reliability with respect to cost. This result is emphasized by the fact that there is a significant difference in how architects prioritize the quality attribute for the cost aspect (COSTUSE).

R12 Regarding what qualities in a software platform that reduce the lead-time in the development of a product, architects in Organization C give no clear answer. System designers have a much more clear opinion as reflected by the significant difference in prioritization related to lead-time (LTUSE). Usability and reusability are the two most important properties. However, when asked what reduces cost the most, system designers claim that maintainability of the software platform is the most important quality. Architects have no clear opinion regarding what quality that reduces cost the most.

R13 Regarding what quality in a software platform that in general causes success in a project that builds on the platform, there are trends that can be interpreted as that both the architect and the system designer consider reliability to be the most important aspect.

3.4 Comments from the participants

This section provides results from interviews and written comments.

R14 In both organizations, the participants are more uniform in their opinions regarding software platform development (Table 4), than in their opinions related to the use of the platforms (Table 5).

R15 When a software platform is bought rather than developed in-house, market dominance and local support of the platform provider are much more important than any of the aspects studied in this survey.

4. Conclusions and applications of results

Several results in this study support a hypothesis that different stakeholders prioritize various quality requirements with impact on the software architecture differently, despite that the organization's goal are the same for the stakeholders. This holds true both for the development of a software platform *(R1, R2, R5, R6 and R7)* as well as the use of a software platform *(R11 and R12)*. This may lead to erroneous balancing of qualities when developing a software platform. It is therefore important to make all the stakeholders understand the prioritization of the quality attributes made for the software platform, since this prioritization may not coincide with the prioritization made by the individual stakeholders. These measures are taken in order to create a software development environment with a good foundation for achieving a mutual understanding of the challenges the different stakeholders are faced with.

In particular, marketing in one of the involved companies believes that achieving reusability is the cheapest quality to achieve. This belief is definitely not shared by the technical stakeholders *(R2)*. This can result in big losses if marketing sells a "reusable" software product too cheap. A risk management process in place as early as during the sales of software that is based on a software platform should take this into account, possibly by involving technical people in the sales process.

Both organizations prioritize reliability as an important aspect in a software platform *(R7, R11 and R13)*. Yet reliability is expensive to achieve in a software platform *(R5)*. This indicates that a software platform development project should not be part of a more market-oriented project, as this can have very hard lead-time requirements. If a plat-form is developed as part of a market-oriented project, it is likely that either the platform does not become reliable, or any other aspect of the product may suffer.

It seems to be better known in the organizations studied what qualities requirements have an impact on cost and lead-time when developing a platform, than what quality in a software platform has an impact on cost and quality when using the platform *(R12, R14)*. This lead us to believe that inadequate metrics for the impact of qualities in a software platform are in use in the organisations. This results in that it is really not known by the stakeholders what quality requirements are most important to focus on when developing a software platform. Therefore, a software development process must ensure that the impact of different qualities are made clear to the stakeholders in software platform development in advance, and a feedback loop from the use of the software platform to the development of the next such product must be in place.

From the above discussion, a few rules of thumb for the development of the architecture of software platforms are devised. These are considered to be generalizable to other large organizations where time to market is a major business driver:

1. In the development of a software platform, reliability has been identified by a multitude of stakeholders to be the most important quality requirement to focus on. Giving one stakeholder, such as an architecture group, the uttermost responsibility for the fulfilment of this quality requirement should emphasize this.

2. A software platform should in general not be developed as a part of another project, or it is likely that either the software platform becomes unreliable due to lead-time constraints, or the project may become late.

3. Since even experienced individuals have varying opinions regarding what is important to have in a software platform, techniques that allow for both the precise communication of what is needed as well as techniques for elicitating these requirements from different stakeholders are needed.

What has not been answered in this study is how a feedback cycle that identifies what quality requirements really affect cost, quality and lead-time in projects that build upon a software platform, can be established. This is subject to further research. It would also be interesting to know why different stakeholders prioritize the aspects as they do, as it could be of value in resolving the conflicts indicated in this study as well as in the teaching of software architects.

The relationship between the quality attributes could be further studied. This could include an analysis of the effects on the other attributes when focusing on improvement with respect to one specific attribute. For example, improvement with respect to one attribute may mean that there, under certain conditions, also will be improvements with respect to the other attributes. Or under other conditions, deterioration with respect to the other attributes.

Finally, extending this study to incorporate more types of stakeholders, e.g. managers, would be interesting since it is hypothesised that this would show further discrepancies in prioritization of qualities in a software platform.

Acknowledgements

This work was partly funded by The Swedish National Board for Industrial and Technical Development (NUTEK), grant 1K1P-97-09673. We thank Prof. C. Wohlin and the Software Engineering Research Group at the Dept. of Communication Systems and Dr. M. Jørgensen and the Industrial Systems Development Research Group at the Department of Informatics, Oslo University, as well as employees at the participating organizations. In particular we thank E. Nordby, I. Rutle and P. Solvang.

References

[1] Bratthall, L., Runeson, P. "A Taxonomy of Orthogonal Properties of Software Architecture". In Proc. Second Nordic Workshop on Software Architecture. Ronneby, Sweden. August, 1999

[2] Bratthall, L., Runeson, P., Adelswärd, K., Eriksson, W. "Lead-time Challenges in the Development and Evolution of Distributed Real-time Systems". Information Systems and Technology. Vol. 14, No. 13, pp. 947-958. September, 2000

[3] Carmone, F.J., Kara, A., Zanakis, S.H., "A Monte Carlo Investigation of Incomplete Pairwise Comparison Matrices in AHP". European Journal of Operational Research, Vol. 102, Issue 3, pp. 538-554. 1997

[4] Cook, T.D., Campbell, D.T. *Quasi-Experimentation - Design and Analysis Issues for Field Settings*. Houghton Mifflin Company. 1979

[5] Datar, S., Jordan, C., Kekre, S., Rajiv, S., Srinivasan, K. "New Product Development Structures and Time-to-Market". Management Science, Vol. 43, No. 4, pp. 452-464. April, 1997

[6] Forman, E., Peniwati, K., "Aggregating Individual Judgements and Priorities with the Analytic Hierachy Process". European Journal of Operational Research, Vol. 108, pp. 165-169. 1998

[7] Frankfort-Nachimias, C., Nachmias, D. *Research Methods in the Social Sciences, Forth Edition*. Edward Arnold, London, Great Brittain. 1992

[8] Graves, S.B. "The Time-Cost Trade-off in Research and Development: A Review". Engineering Costs and Production Economics, Vol. 16, No. 1, pp. 1-9. 1989

[9] Harker, P.T., “Incomplete Pairwise Comparison in the Analytic Hierarchy Process”, Mathematical Modelling”, Vol. 9, No. 11, pp. 837-848. 1987

[10] Harker, P.T., “Alternative Modes of Questioning in the Analytic Hierarchy Process”, Mathematical Modelling, Vol. 9, No. 35, pp. 353-360. 1987

[11] Hayes, W., Over, J.W. “The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers”, Software Engineering Institute, Carnegie Mellon University, USA, CMU/SEI-97-TR-001. December 1997

[12] Hendricks, K.B., Singly, V.R. “Delays in New Product Introduction and the Market Value of the Firm: The Consequences of Being Late to the Market”. Management Science, Vol. 43, No. 4, pp. 422-436. April, 1997

[13] Holm, S. “A Simple Sequentially Rejective Multiple Test Procedure”, Scandinavian Journal of Statistics, Vol. 6, pp. 65-70. 1979

[14] International Standard Organization, *Information Technology - Software Product Evaluation - Quality Characteristics and Guide lines for their Use*, ISO/IEC IS 9126, Geneva, Switzerland. 1991

[15] Karlsson, J., Ryan, K. “A Cost-Value Approach for Prioritizing Requirements”. IEEE Software, Vol. 14, No. 5, pp. 67-74. September/October, 1997

[16] Kruskal, W.H., Wallis, W.A. “Use of Ranks on One Criterion Variance Analysis”. Journal of the American Statistical Association, Vol. 47, pp. 583-621, Corrections appear in Vol. 48. 1952

[17] McCall, J.A., Orchards, P.K., and Waters, G.F, *Factors in Software Quality*, RADC TR-77-369, 1977. Vols. I, II, III, US Rome Air Development Centre Reports NTIS AD/A-049 014, 015, 055. 1977

[18] Miller, R. G. Jr., *Simultaneous Statistical Inference,* Springer-Verlag, New York, USA. 1991

[19] Montgomery, D.C. *Design and Analysis of Experiments, Third Edition*. John Wiley & Sons, New York, USA. 1991

[20] Nagle, T.T., Holder, R.K. *The Strategy and Tactics of Pricing: A Guide to Profitable Decision Making,* Prentice Hall. 1994

[21] Robson, C. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Blackwell Publishers, Great Britain. 1993

[22] Saaty, T.L., *The Analytic Hierarchy Process*, McGraw-Hill, New York, USA. 1980

[23] Schilling, M.A. “Technological Lockout: An Integrative Model of the Economic and Strategic Factors Driving Technology Success and Failure”. Academy of Management Review, Vol. 23, No. 2, pp. 267-284. April, 1998

[24] Stalk, G., Jr. “Time - The Next Source of Competitive Advantage”. Harward Business Review, Vol. 66, No. 4, pp. 41-55. July/August, 1988

[25] Urban, G.L., Carter, T., Gaskin, S., Mucha, Z. “Market Share Rewards to Pioneering Brands: An Empirical Analysis and Strategic Implications”. Management Science, Vol. 32, No. 6, pp. 645-659. June, 1986

[26] Wesslén, A. “A Replicated Empirical Study of the Impact of the PSP on Individual Engineers”, Empirical Software Engineering, Kluwer Academic Publishers, Vol. 5, No. 2, pp. 93-123. 2000

[27] Wheelwright, S.C., Clarc, K.B. *Leading Product Development*. The Free Press, New York, USA. 1995

[28] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. *Experimentation in Software Engineering An Introduction.* Kluwer Academic Publishers. 1999

Paper II:

Is a Design Rationale Vital when Predicting Change Impact?
– A Controlled Experiment on Software Architecture Evolution

Lars Bratthall, Enrico Johansson and Björn Regnell

Proceedings of PROFES 2000, *International Conference on Product Focused Software Process Improvement*, Oulo, Finland, June 2000

Abstract

Software process improvement efforts often seek to shorten development lead-time. A potential means is to facilitate architectural changes by providing a design rationale, i.e. a documentation of why the architecture is built as it is. The hypothesis is that changes will be faster and more correct if such information is available during change impact analysis. This paper presents a controlled experiment where the value of having access to a retrospective design rationale is evaluated both quantitatively and qualitatively. Realistic change tasks are applied by 17 subjects from both industry and academia on two complex systems from the domain of embedded real-time systems. The results from the quantitative analysis show that, for one of the systems, there is a significant improvement in correctness and speed when subjects have access to a design rationale document. In the qualitative analysis, design rationale was considered helpful for speeding up changes and improving correctness. For the other system the results were inconclusive, and further studies are recommended in order to increase the understanding of the role of a design rationale in architectural evolution of software systems.

1. Introduction

Improvement of the software design process may include the introduction of activities related to better and more elaborated documentation of a system's architecture. Typically, architecture documentation describes *what* is present in the architecture in terms of its constituents. It may also be beneficial to spend time on the creation of Design Rationale (DR) documentation, i.e. descriptions of *why* a software architecture is built as it is. A major argument for introducing DR, is its potential benefit as a tool when making changes to the system in the context of architectural evolution. A change task includes the activity of *change impact analysis*, where changed and added architectural components are identified [3], and it can be argued that the rationale for the architecture is vital information when analysing how new or changing requirements impact the design. However, all expansions of the software development process may affect lead-time negatively, as there of course is a cost associated with documenting a system. This cost must obviously be balanced to the potential gain.

This paper presents an empirical study on the effectiveness of DR in software evolution. The main research question is: How effective is DR as a support for change impact analysis? The main motivation for the presented research relates to the general need for a better understanding of how process changes relate to product and process quality. In the presented research, the product quality in focus is *maintainability* and the process quality in focus is *development lead-time*.

The importance of having a short product development lead-time has been argued by many, e.g. [11, 26, 35]. A short product development lead-time increases access to the market introduction window. This in its turn increases the chance of market dominance [34] and decreases the risk of market lockout [28]. Being early on the market also enhances the market's education on a particular product, and increases the likelihood that market survey information still is valid at actual market introduction [31]. Not being late to market is also an issue for shareholders; in a study reported in [16], the average stock value drop was more than five percent when an announced product was later than expected. In [4], a company witnessed that being only one week too late to market resulted in the total annihilation of the potential market, and several million dollars of software investments were wasted. These are examples of an important quality of service that software companies are faced with today: Extremely short time to market or time to customer. Improvements to the software development processes is one way of increasing the development speed.

Short lead-time is, however, not only a requirement for the first member of a product family. For example, in the mobile phone industry, new versions built on existing products have to be release regularly to keep the consumer interest high, and new services that work together with existing systems must be regularly introduced at a high pace.

Maintainability and controlled evolution of a system is dependant on the understanding of what is currently present, as changes in design are affected by the prior design [12]. This understanding is perhaps best acquired through experience in working with the system, or at least through communication access to its original developers. Unfortunately, it is not always possible to communicate with the designers of the original system architecture, or they may have problems recalling the structure of the system. Examples include systems developed with a high staff turnover and systems made by consultants that leave the contractor after the first version of a system has been accepted. For these systems, it is desirable to establish communication which is available over time, and a documentation of DR is, in this situation, assumed to be important for the understanding of the present system architecture.

The problems that occur when changes are being made to poorly understood problems have been highlighted by e.g. [5, 7]. An important problem is architectural erosion [25]: A system that is being changed when the architecture has not been understood erodes into an entity where new change requests are becoming harder and harder to fulfil, and eventually a change request is either impossible to accommodate, or it results in more new errors than those potentially being fixed. This is a serious problem in domains where lead-time is an important issue, since lead-time accelerating activities such as product-family or product-line reuse requires a strong grip of the architecture. In the light of these arguments, we have designed a controlled experiment to investigate the hypothesis of DR documentation being an effective means for supporting change impact analysis.

The paper is structured as follows. Section 2 gives a brief description of the selected approach to design rationale documentation. Section 3 explains the variables and hypothesis being studied, as well as the design of the experiment. In Section 4 the data from the experiment is analysed, and in Section 5 the results are interpreted in relation to the hypothesis of the experiment together with conclusions and issues of further research.

2. Object of study: A specific approach to design rationale documentation

DR was early suggested as an important part of software architecture [25], and in [1] the importance of documenting DR is also recognized. According, to Shum [29], DR research originates from argumentation formalisms (e.g. [32, 33]), hypertext representational technology [8, 14, 24], design research [10] and software engineering. This paper focuses on software engineering aspects, as it investigates the potential business value of DR in an actual software design environment.

A DR is a kind of documentation, that not necessarily must be produced before the first release of a system. Instead, it can be written after the development of the first generation of a system as a retrospective DR [29], albeit with the risk that the DR is biased by the person writing the DR [21]. A retrospective DR does not delay the market introduction of the first generation of a system, as does a narrative DR [29], written during the initial development.

There are many approaches to both representing and manipulating a DR [17, 20]. A weakness in some approaches to DR, e.g. QOC [29], rIBIS [27], is that they may need substantial training in order to be used effectively. Another weakness is the weak association between a DR document and the actual code. This is a problem, since designers usually dislike making additional documentation [29]. Also, it has been noted that some kind of style guide is necessary if a good DR should be produced, and a structure of DR has been asked for in [29].

Our approach to DR is very simple, yet it tries to address the previously discussed deficiencies: For each aggregation level [6] in a system, the original designer should write a short note on why it is broken down into the lower aggregation levels, and what the purpose of each component is. This ensures that the DR has a strong association with the code, and that the DR's size is limited which is not the case for e.g. a QOC DR [18]. On a system level, comments should be made on only four standardized headings: Organization of system into files, use of language constructs, main dynamic architectural principles and finally, clues to understanding the system. Together, the DR provides explicit information for Kruchten's [19] development view and implicit information on any of the logical, the synchronization, and the physical views. The value of structuring the DR according to the 4+1 View Model [19] is that its information content is field-proven to be useful in architecture level design. The reason why architectural level design is addressed is that during this level of design, decisions with system-wide implications must be taken, which both constrain a system's natural change room, as well as facilitates changes within this room. Thus an architectural understanding is crucial for maintaining flexibility in a system during its evolution.

The suggested approach has several strengths. First of all, it requires only little training. The writing of a DR for a component at a particular aggregation level can be prompted by a code entry tool, resulting in higher likelihood of actually producing the documentation. Since the documentation follows the structure of the code, the likelihood of the appropriate DR being found when changes are being performed increases. Since the DR documentation is tightly associated with the code, the likelihood that it will be maintained when code changes increases, as a designer does not have to go through the a tedious of updating multiple documents. Apart from this, the approach scales well also to large systems, unlike e.g. QOC [18], and can easily be used with many existing case tools which is important since those generally lack the explicit ability to capture design rationale at different levels of abstraction [13].

In summary, the chosen DR approach which is the object of this empirical study includes natural language explanations of (1) the rationale for the static structure of the system; (2) the organization of the system into files; and (3) rationale for other design decisions considered important, when the original designers subjectively thinks about future likely product changes.

3. Experiment planning and operation

The experiment is motivated by the need to reduce the time spent in changing complex systems when it is not possible to communicate with the original developers, or the original developers have forgotten much of the system at hand. The aim of this study is to evaluate the effectiveness of having access to a simple textual design description, where care has been taken to motivate why the system is designed in the way it is, i.e. a DR is available.

The experiment is defined as follows:

- Object of study: A textually represented DR combined with some system static architectural description [6] at various aggregation levels [6].
- Purpose: The effectiveness of a DR is studied. Effectiveness is measured in terms of how well change requests are being performed, as well as how long time these change requests take to fulfil.
- Perspective: Single experienced developers coming to a new system, without previous knowledge of it.
- Context: The effectiveness is measured when attempting changes to two real-time systems: A local telephony switch control software and a car cruise control system. Participants are from two groups: Senior industrial designers and a group of Ph.D. students and faculty members in software engineering.

3.1 Variables and hypotheses

The independent variables are variables that we can control and manipulate [37]. There are two independent variables: The provision of a DR – or not, and the system where change requests are applied on. Our main hypothesis is that people design differently when having access to a DR. Three more formal hypotheses are shown in Table 1. All hypotheses are tested at $p \leq 0.10$ on a Windows PC running SPSS version 9.0. The level of significance is chosen beforehand to avoid fishing for particular results.

Table 1. Hypotheses. $Sys \in \{\text{System A, System B}\}$

Hypothesis
$H_{t,\,Sys,\,1}: \overline{t_{Change,\,Sys,\,WithDR}} \neq \overline{t_{Change,\,Sys,\,NoDR}}$ There is a difference in how long time it takes to complete the change tasks depending on whether a DR is available or not. Null hypothesis: There is no difference in this aspect.
$H_{PercOK,\,Sys,\,1}: \overline{PercOK_{Sys,\,WithDR}} \neq \overline{PercOK_{Sys,\,NoDR}}$ There is a difference in how large percentage of the required changes are correctly suggested when a DR is available and when it is not. Null hypothesis: There is no difference in this aspect.
$H_{NoExtra,\,Sys,\,1}: \overline{NoExtra_{Sys,\,WithDR}} \neq \overline{NoExtra_{Sys,\,NoDR}}$ There is a difference in how many superfluous, incorrect or unsuitable changes are suggested when a DR is available and when it is not. Null hypothesis: There is no difference in this aspect.

3.2 Design

There are 17 participants in the experiment of which 7 are industrial senior designers active in a designated software architecture group. The rest are faculty members or Ph.D. students with varying industrial experience. All participants have received training in the formality of the source models used, the form of the change requests, and how to indicate where they believe that changes are requested in order to fulfil each change request. The participants have been exposed to two or three systems, and a number of change requests for each system. One group of participants has had access to a DR, and the other group has not.

Before the experiment, all participants filled in a form describing their knowledge and experience. In total, 17 aspects of their experience and knowledge have been asked for. The experience from real-time systems and the modelling language have been used to randomize participants regarding their access or no access to a DR. The reason that these two aspects have been used to randomize participants is that these aspects have been seen in a prior experiment to have a relatively large correlation with the ability to quickly identify errors in distributed real-time systems [2] of similar complexity as those studied in this experiment.

In the beginning of the experiment, an introduction and an overview of the model language used to describe code, SDL [30], were given. SDL describes the code graphically using extended finite state machines, which can be hierarchically grouped. The introduction was guided by slides. These slides were available to all participants as handouts. These handouts were used as a reference when the code was studied, in case something in the modelling language should be unclear. Through the introduction and the model language overview, it was made sure that all participants had a reasonable understanding of the code modelling language and initial learning effects would not affect the experiment. It was also made certain that everyone knew what to do and how to fill in the forms.

All participants were first assigned four change requests in random order, that required changes to a local telephony system – system A (described in Table 2). An example change task is shown in Figure 1. The change requests are realistic requests for new services. The author writing the DRs has had no knowledge of the change tasks to come, thus reflecting reality. This seem to differentiate this study from e.g. [15] and several studies in [29].

Purpose: The system must be updated in order to maintain speed and safe distance from other vehicles. A highly sensitive radar will be used to precisely locate the position of the cars ahead and behind. The cruise control should be able to take advantage of this new functionality. All the described functionality must be implemented.

Description: Detailed description (e.g. it must be possible to set the distance using particular switches)

Constraints: Constraints (e.g. The solutions must use a particular display present in the car).

Figure 1. *Example of change task*

Table 2. Description of systems

System A is a real-time system that controls the operation of a local PBX. The software is based on asynchronous processes, and it is modelled in SDL [30], which is a high-level well-defined language. The requirements specification is rather complete and well-written. The system has previously been described in [36]. The system is described at 3 aggregation levels containing seven different static software processes. The maximum number of concurrently executing software processes is 28.
System B is a real-time system written in SDL that consists of two distinct parts: i) A car cruise control and ii) a test-driver stub for i). The test-driver allows the simulation of a road and driver actions while allowing monitoring of speed and dash-board indicators. The system is in industrial use. The requirements are described on less than a half page of text. This specification is incorrect as well as incomplete – a too common situation in an industrial context. The system is described at 5 aggregation levels containing 18 different static software processes. The maximum number of concurrently executing software processes is 18.
System C is a building access control real-time system written in SDL that consists of two distinct parts: i) A part that is a central database for key-cards and their authorization and ii) A part that is the real-time handler for each door. The requirements are described on four pages, and the requirements are considered as being correct and complete. Their is also a short definition of the vocabulary used. The system is described at 3 aggregation levels containing 2 different static software processes. The maximum number of concurrently executing software processes is potentially unlimited, depending on the number of doors connected to the system. In the program provided to the participants, there are 2 concurrently executing software processes.

For practical reasons, the maximum time allowed for each change request was nine minutes, not including maximally 5 minutes for studying the extent of documentation and reading each change task. The suggested change impact is recorded in a form where it is possible to indicate components requiring internal change, as well as addition of new components at various aggregation levels. Parts of such a form is illustrated in Figure 2.

Indicate where you believe you will have to make changes for the current change task here.		
What (Components in software)	**An X indicates that you believe you would like to make a change**	**An X indicates that you would like to add a SDL-process or an SDL-block in this block**
Cruise_Requirements Analysis		

Figure 2. *Parts of form for recording where changes are likely, i.e. "change points"*

After the change tasks related to system A, the participants have been exposed to either of two experimental designs. The reason for this is that it was seen that the first pilot experimental design did not work well – in short, there was so little time available for the change requests that the participants did not succeed at all in delivering answers to the change requests.

Design I (pilot): After the telephony switch control system, both a car cruise control system and a building access control system was provided in random order, together with four change requests for each system, with the change requests for each system in random order. The time limit for each change request was maximized to nine minutes for practical reasons. In practise, this limit proved to be far to low for the cruise control system (system B, described in Table 2), and the building access control (system C) proved to be so simple that almost all participants gave equal answers. Therefore, only data from the telephony switch control systems are retained and analysed in this paper.

Design II (main run). After system A had been addressed, all participants were faced with the cruise control system. Only two change requests were provided in random order. These change tasks were sampled from a commercial patents database. The time-limit for these two change requests were maximized to thirty minutes each. In practise, this design worked well, since there was ample time to comprehend the cruise control system.

Finally, after completing all change tasks, the participants were interviewed to get some subjective data. The interviewer used a predefined order of asking defined questions, i.e. schedule-structured interviews have been performed. This kind of interview ensures that variations in answers to as large extent as possible are attributable to the respondents and not from variations in the interviews [23].

Each change task is compared to a system expert written solution, containing required change-points (both additions and changes in Figure 2). Indicated changes that are not part of the expert solution are called superfluous change points.

3.3 Threats to validity

The validity of the findings is highly dependent on how well threats have been handled. Four types of validity threats [9] are analysed: Threats to conclusion validity, construct validity, internal validity, and external validity. The relationships between these are illustrated in Figure 2.

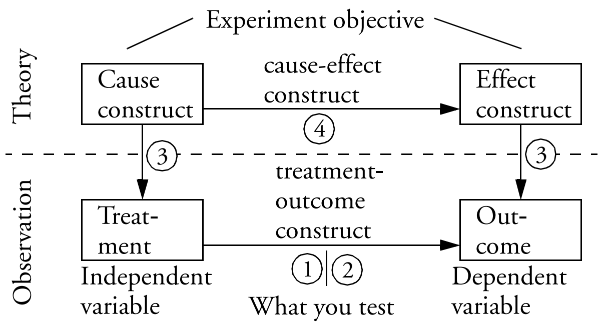

Figure 2. *Experiment principles as described in [37]*

Conclusion validity. Conclusion validity (marked 1 in Figure 2) concerns the relationship between the treatment and the outcome. Care has been taken not to violate any assumptions made by the statistical tests. All time measurements have minutes as their unit, and the participants have been instructed to use the same clock during all measurements. Combined with the fact that there is no gain for a participant in adjusting their measurements, the reliability of measures should be good. There is a risk that participants have used the supplied DR in different ways, but this threat has been addressed by explicitly allowing time for studying the available documentation for each system, before the change tasks were delivered to the participants. There are no random irrelevancies (such as mobile phone calls during the experiment) that we are aware of. The threat of a large random heterogeneity of subjects have been balanced by proper randomization of participants and treatments. Each participant has been assigned individually to a group firstly depending on their self-assessed knowledge of real-time systems, secondly depending on their knowledge of system A (some participants have seen this system before). To the best of our beliefs, the individual performance has been cancelled out, as has the effect of learning the systems during the experiment.

Internal validity (2). This validity concerns matters that may affect an independent variable's causality, without the knowledge of the researcher. Maturation effects have been countered by making sure that the experiment does not take more than two and a half hour to conduct. The random order of change requests should cancel any other maturation and learning effects. There may be selection effects present, since the participants have been unusually well educated or well-experienced designers. However, we do not think that this affects the results other than that the difference between the group that had access to DR and the other group, without DR, may be smaller than in a less experienced group of participants. There has been no mortality during the experiment. Since the experiment was given with and without DR at the same time, the control group (without DR) cannot have learned about the treatment (access to DR) in advance, and thus cannot have imitated the solution strategies used by the DR-equipped group.

Construct validity (3). Construct validity concerns whether we measure what we believe we measure. There are two main threats to construct validity: Design threats and social threats.

As there are several change tasks and two different systems, the effect of peculiarities of a single system/change task should be low. We know of no confounding level of constructs, and the participants have been randomized as to cancel any effect of having different knowledge and experience a priori the experiment. Given the careful presentation of the experiment and the full anonymity provided, there should be no social threats.

External validity (4). This last validity concerns generalization of the findings to other contexts and environments than the one studied. There may be some problem with generalizing the results to less experienced groups, such as students going directly from university to industry, since all participants are either very well-educated or have a senior designer experience level. However, we believe that the difference between the group receiving DR and the control group, without DR, should be bigger in a group with less experienced designers. The change tasks are considered realistic for system A and very realistic for system B since the change tasks are sampled from a patents database. Thus they should be representative in an industrial environment. System B as well as its accompanying documentation are industrially operational system, while system A is an educational system, but of such size and complexity that we believe it to be comparable to industrial systems.

In summary, great care has been taken in the design of the experiment, so the threats to validity should be under control.

4. Data analysis

The purpose of this section is to present results and statistical analysis of data collected. The experiment provides quantified data. The hypotheses have been tested using the non-parametric Mann-Whitney test, at $p \leq 0.10$. The results are presented in Sections 4.1 and 4.2. The subjective data from interviews are investigated in Section 4.3.

4.1 Analysis of experiment data, system A

For the change-tasks for system A, there is a statistically significant difference in both the time spent on the change tasks as well as the quality of the predicted changes, based on data from 57 completed change tasks. All data are illustrated in Figures 3-5, and the results of the statistic analyses are summarized in Table 3. No participants are treated as outliers in the analysis. The boxplots all show an improvement (Figures 3 and 4) or at least no clear difference (Figure 5) between the group having access to a DR, and the group that does not. The statistic tests reflect this, by rejecting the null hypotheses $H_{t,\, SysA,\, 0}$ and $H_{PercOK,\, SysA,\, 0}$.

The null hypothesis $H_{NoExtra,\, SysA,\, 0}$ is not rejected. This is interpreted as that there is no clear difference between the two groups in this aspect. This interpretation is strengthened from Figure 5.

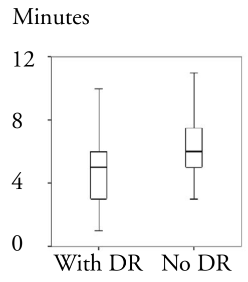

Figure 3. *System A, time used for change tasks*

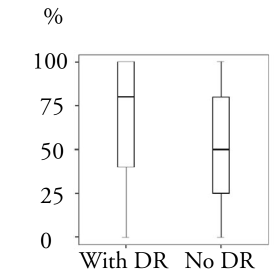

Figure 4. *System A, percentage of required change points*

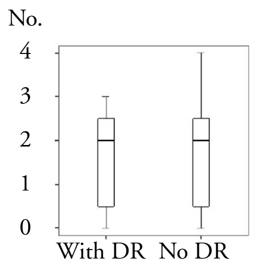

Figure 5. *System A, number of superfluous change points*

Table 3. Summary of statistical analysis at significance level $p = 0.10$

	Sys = System A		Sys = System B	
	Mann-Whitney	**Illustration**	**Mann-Whitney**	**Illustration**
$H_{t,\,Sys,\,0}$	Reject	Figure 3	No reject	Figure 6
$H_{PercOK,\,Sys,\,0}$	Reject	Figure 4	No reject	Figure 7
$H_{NoExtra,\,Sys,\,0}$	No reject	Figure 5	No reject	Figure 8

4.2 Analysis of experiment data, system B

For the change-tasks for system B, there is a no statistically significant difference in neither the time, nor any of the quality measurements taken for the change tasks. This is based on data from 20 completed change tasks. All data are illustrated in Figures 6-8, and the results of the statistic analyses are summarized in Table 3. No participants are treated as outliers in the analysis. Judging from the medians in the figures, there is a trend showing that it is beneficial to have access to a DR. For example, the median time for accomplishing each change task decreases from 20 minutes to less than 15 minutes, while the median of correctness in answers increases. These results are similar to those from system A, which strengthens the position that a DR can be beneficial. However, the Mann-Whitney test does not detect any significant difference in any case tested.

It is possible that the statistical tests cannot detect a significant difference between the two groups, given the lower number of data-points (20), and rather small difference between the groups. Therefore, the results are inconclusive for this system.

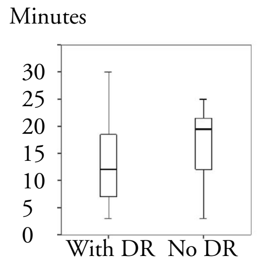

Figure 6. *System B, time used for change tasks*

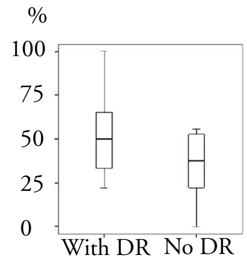

Figure 7. *System B, percentage of required change points*

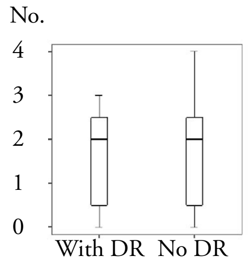

Figure 8. *System B, number of superfluous change points*

4.3 Analysis of interview data

This section presents some subjective data elicited during the interviews. One of the questions asked was “How much faster can you solve the change task (comfortably well) with access to a DR?”. The results are presented in Figures 9 and 10. The participants believe that there is some improvement in development lead-time with access to a DR for the less complex system A, and a high degree of improvement for the more complex system B.

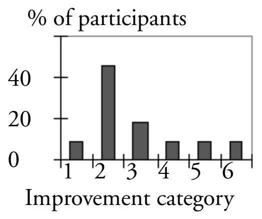

Figure 9. *System A, change in lead-time with access to DR*

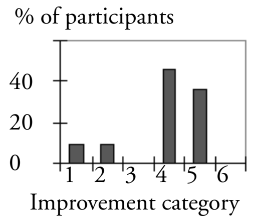

Figure 10. *System B, change in lead-time with access to DR*

	Improvement categories
1	No opinion
2	0-19% faster with DR
3	20-39% faster with DR
4	40-59% faster with DR
5	60-79% faster with DR
6	80+ % faster with DR

Another question was “To what degree do you think that a DR increases your correctness in change predictions?” with results presented in Figures 11 and 12. The participants indicate no or a little improvement for the less complex system A, and a much higher degree of improvement for the more complex system B.

It should be noted that the participants appreciated the DR more for the complex system B than for the less complex system A. No participants claimed that having access to a DR was harmful. Several participants witnessed that they believe that the effectiveness of the DR decreases as the system at hand gets better known.

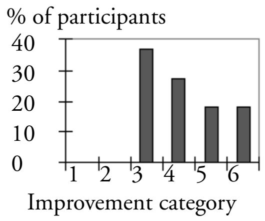

Figure 11. *System A, change in impact prediction correctness with access to DR*

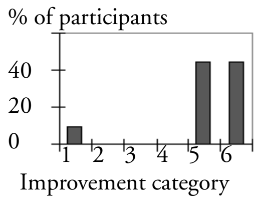

Figure 12. *System B, change in impact prediction correctness with access to DR*

Improvement categories	
1	No opinion
2	It makes predictions worse
3	It does not affect correctness at all
4	I become marginally more correct
5	I become more correct
6	I become a lot more correct

5. Summary and conclusions

This section gives an interpretation of the results presented.

Regarding system A, it did take significantly shorter time for the participants to accomplish the change-tasks when having access to a DR, and the quality of the results were significantly better or possibly equal than for the group that did not have access to a DR. These objective results are further reinforced by the subjective interview data.

Regarding system B, the picture is not as clear. The median time used for the change tasks is shorter for the group having access to DR than the non-DR group. The median percentage of correctly indicated change points is also better for the DR-group. However, this result is not statistically significant. There may be many reasons for this, such as unrealistic experimental procedures, too few data points (there are much fewer data points for system B, since we discarded all data related to this system from the pilot run of this experiment), or that qualities in the system itself affect the effectiveness of having access to a DR. Regardless, it calls for further analysis.

Information from the interviews suggests that the participants liked having access to a DR. They believed that could work both faster and better, and a Mann-Whitney test shows that the group that had access to a DR believed they would need significantly ($p \leq 0.10$) shorter time to solve the change-tasks related to system B than the group without a DR. All participants believe that they work faster and better with access to a DR, than when no DR is available, when no other documentation is available than the source code and the requirements specification.

In short, we conclude the following and suggest some future lines of work:

It is likely that having access to a DR expressed in the suggested way have a positive impact on both lead-time as well as quality when experienced designers are faced with the task of predicting where changes must be performed on an unknown real-time system. However, it is possible that there are better ways of achieving the same results using other models or ways of transferring the sought for knowledge to maintainers. For example, participants frequently indicated that they needed a static architecture overview and sequence diagrams/MSCs. Further experimentation is needed to find out what is "the best" model for various purposes.

In projects where lead-time is important it is possible that a sharp schedule prohibits the creation of models during design. In that case, writing a DR in the suggested manner after initial system release may be a cheap, yet effective way to facilitate future system evolution, without prolonging the time to initial system release. This result can easily be incorporated in standard development processes.

Acknowledgements

This work was partly funded by The Swedish National Board for Industrial and Technical Development (NUTEK), grant 1K1P-97-09690. We thank the designers at Ericsson Mobile Communication AB, Lund, Sweden for interest and participation in this study. Prof. Claes Wohlin at the Dept. of Communication Systems, Software Engineering Research Group, Lund University has given insightful comments on this paper, as well as Dr. Magne Jørgensen at the Dept. of Informatics, Industrial Systems Development Group, Oslo University. Lars Bratthall is currently at Oslo University.

References

[1] Bass, L., Clements, P., Kazman, R. *Software Architecture in Practise.* Addison Wesley. 1998

[2] Bauer, N., Olsson, T., Runeson, P., Bratthall, L. "Lead-time Impact of Distributed Testing on Distributed Real-time Systems". To be published in Proc 7th International Metrics Symposium (METRICS 2001). London, England. April, 2001

[3] Bohner, S., Arnold, R. (Eds). *Software Change Impact Analysis*. IEEE Computer Society Press. 1996

[4] Bratthall, L., Adelswärd, K., Eriksson, W., Runeson, P. “A Survey of Lead-Time Challenges in the Development and Evolution of Distributed Real-Time Systems”. Information and Software Technology. Vol. 42, No. 13, pp. 947-958. 2000

[5] Bratthall, L., Runeson, P. “Architecture Design Recovery of a Family of Embedded Software Systems - An Experience Report”. In Proc. TC2 First IFIP Working Conf. on Software Architecture, pp. 3-14. San Antonio, Texas. February, 1999

[6] Bratthall, L., Runeson, P. “A Taxonomy of Orthogonal Properties of Software Architectures”. Proc. 2nd Nordic Software Architecture Workshop. Ronneby. August, 1999

[7] Brooks, F. *The Mythical Man-Month: Essays on Software Engineering.* Ingram Int’l., USA. 1995

[8] Conklin, J. “Hypertext: An Introduction and Survey”. IEEE Computer, Vol. 20, No. 9, pp. 17-41. 1987

[9] Cook, T.D., Campbell, D.T. *Quasi-Experimentation – Design and Analysis Issues for Field Settings*. Houghton Mifflin Company. 1979

[10] Cross, N. “The Nature and Nurture of Design Ability”. Design Studies, Vol. 11, No. 3, pp. 127-140. 1990

[11] Datar, S., Jordan, C., Kekre, S., Rajiv, S., Srinivasan, K. “New Product Development Structures and Time To Market”. Management Science, Vol. 43, No. 4, pp. 452-464. April, 1997

[12] Fisher, G., Lemke, A.C., McCall, R., Morch, A.I. “Making Argumentation Serve Design”. Human-Computer Interaction, Vol. 6, No:s 3&4, pp. 393-419. 1991

[13] Grundy, J. “Software Architecture Modelling, Analysis and Implementation with SoftArch”. proc. 34th Hawaii International Conference on System Sciences. January, 2001

[14] Halasz, F.G. “Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems”. Comm. of the ACM, 31, pp. 836-852. 1988

[15] Hamilton, F., Johnson, H. “An Empricial Study in Using Linked Documentation to Assist Software Maintenance”. In Human-Computer Interaction (Interact ‘95), pp. 219-224. Chapman & Hall, London. 1995

[16] Hendricks, K.B., Singhal, V.R. “Delays in New Product Introduction and the Market Value of the Firm: The Consequences of Being Late to the Market”. Management Science, Vol. 43, No. 4, pp. 422-436. April, 1997

[17] Jarczyk, A.P.J., Løffler, P., Shipman III, F.M. “Design Rationale for Software Engineering: A Survey”. In Proc. 25th Annual Hawaii Int’l Conf. on System Sciences. pp. 577-586. 1992

[18] Jørgensen, A.H., Aboulafia, A. “Perceptions of Design Rationale”. In Human-Computer Interaction (Interact ‘95), pp. 61-66. Chapman & Hall, London. 1995

[19] Kruchten, P.B. “The 4+1 View Model”. IEEE Software Vol. 12, No. 6, pp. 42-50. 1995

[20] Lee, J., Lai, K.”What’s in Design Rationale?”. Human-Computer Interaction. Vol. 6, No.s 3&4, pp. 251-280. 1991

[21] Lee, J. “Design Rationale Systems: Understanding the Issues”. IEEE Expert, Vol. 12, No. 3, pp. 78-85. May/June, 1997

[22] Message Sequence Charts (MSC), ITU-T Standard Z.120. International Telecommunication Union. 1996

[23] Frankfort-Nachmias, C., Nachmias, D. *Research Methods in the Social Sciences, Fourth Edition.* St. Martin’s Press, United Kingdom. 1992

[24] Nelson, T.H. “A File Structure for the Complex, the Changing, and the Indeterminate”. Proc. ACM National Conference. pp. 84-100. 1965

[25] Perry, D.E., Wolf, A.L. “Foundations for the Study of Software Architecture”. Software Engineering Notes. Vol. 17, No. 4, pp. 40-52. October, 1992

[26] Porter, M.E. *Competitive Strategy - Techniques for Analyzing Industries and Competitors.* The Free Press, New York, USA. 1980

[27] Rein, G.L., Ellis, C.A. “rIBIS: A Real-time Group Hypertext System”. Int’l. Journal of Man-Machine Studies, Vol. 34, No. 3, pp. 349-367. March, 1991

[28] Shilling, M.A. “Technological Lockout: An Integrative Model of the Economic and Strategic Factors Driving Technology Success and Failure”. Academy of Management Review, Vol. 23, No. 2, pp. 267-284. 1998

[29] Shum, S.J. *A Cognitive Analysis of Design Rationale Representation.* Ph.D. Thesis, York University, Great Brittain. December, 1991

[30] Specification and Description Language (SDL), ITU-T Standard Z.100. International Telecommunication Union. 1992

[31] Stalk, G. “Time – the Next Source of Competitive Advatage”. Harward Business Review, Vol. 66, No. 4, pp. 41-55. 1998

[32] Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., Suchman, L. “Beyond the chalkboard: Computer Support for collaboration and problem solving in meetings”. Comm. of the ACM, Vol. 30, No. 1, pp. 32-47. 1987

[33] Toulmin, S. *The Uses of Argument.* Cambridge University Press. Cambridge, Great Brittain. 1958

[34] Urban, G.L., Carter, T., Gaskin, S., Mucha, Z. “Market Share Rewards to Pioneering Brands: An Empirical Analysis and Strategic Implications”. Management Science, Vol. 32, No. 6. June, 1986

[35] Wheelwrigt, S.C., Clark, K.B. *Leading Product Development - The Senior Manager's Guide to Creating and Shaping the Enterprise.* The Free Press, New York, USA. 1995

[36] Wohlin, C. "The Challenge of Large Scale Software Development in an Educational Environment". Proc. Conf. on Software Engineering Education & Training, pp. 40-52. Virginia Beach, Virginia, USA. 1997

[37] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston, MA, USA. 1999

PAPER III:

Benchmarking of Processes for Managing Product Platforms - a Case Study

Martin Höst, Enrico Johansson, Adam Norén and Lars Bratthall

Proceedings of Empirical Assessment in Software Engineering, (EASE 2002), Keele, England, April, 2002

Abstract

This paper presents a case study where two organisations participate in a benchmarking initiative in order to find improvement suggestions for their processes for managing product platforms. The initiative is based on an instrument which consists of a list of questions. It has been developed as part of this study and it contains eight major categories of questions that guide the participating organisations to describe their processes. The descriptions are then reviewed by the organisations cross-wise in order to identify areas for improvement. The major objective of the case study is to evaluate the benchmarking procedure and instrument in practice. The result is that the benchmarking procedure with the benchmarking instrument was well received in the study. We can therefore conclude that the approach probably is applicable for other similar organisations as well.

1. Introduction

Effective management is important for the process of developing platforms on which a number of products are based and for the process of developing the platforms themselves. A platform can be seen as the core asset of a product line architecture, a type of architecture which is used in products that come in many versions or families. These families are based on the same platform to reduce cost and shorten lead-time [1]. Software companies should not have to re-create the infrastructure for each new project, instead the development and the management activities should be based on well-defined processes and a solid product architecture. It is also critical that managers, platform developers and product developers all share the same vision for a software platform.

"Technically excellent product line architectures do fail, often because they are not effectively used. Some are developed but never used or, if used, they are used in an incorrect way." [2] There is a large number of users of a platform (e.g. software product developers), and there can be several different concurrent product projects using the platform. Management processes are needed to support dependent development efforts in different phases of product projects, platform projects and across the separate development teams involved.

Processes for managing product platforms (Product Platform Management Processes, PPMP) have recently been given much attention, see for example [3, 4, 5]. In practice, there are of course many different processes in place, which means that companies constantly have to evaluate and improve their processes in this area. The objectives for an organisation for carrying out a study as the one presented here are to identify improvement proposals for their PPMP and, in the long run, to introduce improvements in the process. That is, this kind of study can be part of a software process improvement program.

In this paper a procedure for carrying out a benchmarking initiative in order to improve PPMPs is presented and evaluated in a case study. In Section 2 the benchmarking procedure is described and the case study is presented in Section 3. In Section 4 the conclusions from the study are reported.

2. Benchmarking methodology

2.1 Introduction

Software process improvement is important for all software development and management processes. Often the steps that are taken in software improvement is to first carry out an assessment, and then, based on the assessment, identify improvement proposals and after that to introduce, evaluate, and tune the changes [6]. The benchmarking technique that is presented here is one technique that can be used for assessment and identification of improvement proposals.

Benchmarking [7, 8] has been used as a general improvement approach in a variety of business areas. The basic idea behind the benchmarking concept is that for each company, there are a number of other companies that have been working with the same issues and problems. A company that wants to improve in a certain area should therefore identify appropriate companies to compare themselves with and learn from. It is, of course, important that competitors are not chosen. Instead, companies which are working in the same way with similar processes, but with products not competing, should be chosen. In the study that is presented in this paper, the companies have in common that they are developing new products based on a common platform, and that they have a PPMP.

Camp presents a benchmarking process using four major steps [8]. According to that process the first step is *planning*, where it is decided what business function or type of product the benchmarking initiative should focus on, what companies should be involved in the initiative and details on how the initiative should be carried out. For example, it is decided how data should be collected during the work. The second step is the *analysis* step, where the actual comparison of the companies is carried out. The third step is an *integration* step, where it is decided and planned how findings from the initiative can be integrated in the current process. It is, for example, important to obtain management and operational acceptance and to communicate the changes to all levels of the organisation. The fourth step is the *action* step, where the changes are implemented and the results are monitored.

The benchmarking approach that is presented and evaluated in this paper concerns the first two steps (planning and analysis) of the process presented above. It is also tailored to be used for comparing PPMPs. That is, it gives guidance on how to compare two processes for product platform management. After the comparison it is important to identify improvement actions from the result of the comparison and to actually implement the changes. It would, however, be too much to cover implementation issues in this paper.

There are a number of issues from the philosophy of Total Quality Management (TQM) that are relevant in the area of benchmarking. Benchmarking initiatives can be designed around the key issues of, for example, the Malcolm Baldridge approach [9] or the Capability Maturity

Model from the Software Engineering Institute [10]. Part of the approach that is presented here is influenced by the method of performing quality evaluations according to, for example, the Malcolm Baldridge approach. However, in this approach the involved organisations are not compared to any general model. Instead, they are compared to each other in areas that are of interest for them.

Organisations can be analysed with respect to a number of dimensions [11, 12], e.g., with respect to the *approach* (i.e. the process) that they describe, the *deployment* of the approach, i.e., to which extent the work according to the approach is performed in the organisation, and the *results* that the approach results in.

The organisations that participate in this kind of initiative have a number of objectives for participating. First of all, one objective is to receive information on how another company has organised their work in the area. That is, it is a way of receiving improvement proposals from the work of another organisation. Another objective is to receive feedback on their processes from experts in the area. The idea of the approach is that the participating organizations should achieve their objectives by participating in the work. That is, if two organizations participate, both of them should reach their goals by mutually analysing and commenting on the processes of each other.

2.2 Benchmarking process

In this section, a benchmarking approach that can be used for analysing PPMPs is presented. The benchmarking approach has been evaluated in a case study, which is presented in Section 3. The benchmarking process is based on a Benchmarking Instrument (BI), which is a list of questions that the participating companies use to develop descriptions of their PPMPs. The following benchmarking process can be used in a benchmarking activity with two organisations (A and B):

1. Identify important issues to work with and agree on a final BI. This may, for example, be done by letting the participating organisations review an initial proposal for a BI. In this way they will be able to add issues that they are interested in and remove issues that they are not interested in. The final BI that was used in the case study is presented in Section 2.2.12.2.1.

2. Each organisation develops a presentation of their PPMP according to the BI. This results in one document from each company (P_A, P_B).

3. Each organisation reviews the presentation written by the other company. The reviews are carried out according to a review template (RT) which is presented in Section 2.2.3. Each review results in a number of comments to the presentation (CP). That is, Company A reviews P_B, which results in CP_B, and Company B reviews P_A, which results in CP_A. The comments to the presentations are sent to the benchmarking coordinator.

4. CP_A is distributed to Company A and CP_B is distributed to Company B.

5. A meeting is held with representatives from each company. At this meeting, it is, for example, possible to discuss common improvement proposals.

The procedure may be organised by a coordinator who is responsible for initiating the different tasks and delivering the needed documentation etc. This is illustrated in Figure 1. The role of the coordinator is refined in Section 3, where the case study is presented. Step 2 of the benchmarking approach is further described in Section 2.2.2, and Step 3 is further described in Section 2.2.3.

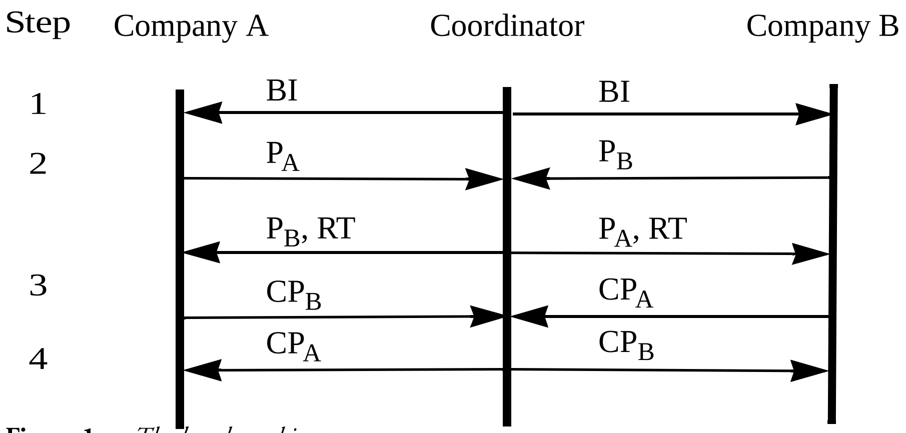

Figure 1. *The benchmarking process.*

The idea of carrying out a benchmarking initiative by letting organizations review descriptions of each other is not unique to this approach. The benchmarking instrument that was used is however developed especially for this study.

2.2.1 Benchmarking instrument

In this section, the benchmarking instrument that has been used is presented. The benchmarking instrument consists of a number of questions that are presented in Table 1. There are questions at two levels, one high-level and one more detailed level. For every high-level question there are a number of detailed questions. The benchmarking instrument guides the participating organisations to describe their PPMP with respect to the following issues:

- Introduction/Context. In order to better interpret answers to the other questions in the questionnaire, there are a number of questions that characterize the development organization in general. The participants are asked to make an overview of the architecture of their product, as well as of their development process. In this way, it is possible to interpret answers in the context of, e.g., evolutionary development or more waterfall-model development. These questions are not further analysed in this paper and they are not included in Table 1.
- Organizational context for architecture work. The participating organizations are known to have a group that is responsible for their product architecture. Questions Q1 and Q2 concern the responsible groups' empowerment and responsibilities.
- Initial architecture design. This section contains detailed questions regarding architecture development. The questions in this section are intentionally formulated open-ended, as it is believed that this will help elicit a broad spectrum of techniques. This is mainly covered by questions Q3, Q4, and Q5.
- Architecture use and evolution. At some point, the architecture is no longer not only the property of an architecture group as many people must use it. This section focuses on the deployment and follow up of the architecture. This is mainly covered by questions Q6, Q7, and Q8.

Table 1. Questions in benchmarking instrumentation.

Id	Question
Q1	How is the responsibility and empowerment for the group responsible for the platform defined?
Q1a	How and by whom is the group responsible for the platform constituted?
Q1b	Who decides when the group responsible for the platform should be involved in the decision-making?
Q1c	What and who gives the group responsible for the platform the ability to perform?
Q2	Describe the stakeholder interaction.
Q2a	To which stakeholders do the group responsible for the platform communicate?
Q2b	What commitments are made to different stakeholders?
Q3	Which procedures are used to identify architectural requirements?
Q3a	What different types of architectural requirement are identified?
Q3b	How are the different types of requirements handled?
Q3c	How are the architectural requirements prioritized?
Q4	What design process is used?
Q4a	What methods are used when identifying the requirements impact on the architecture?
Q4b	What languages are used in order to design the architecture?
Q4c	What granularity does the architecture design cover?
Q4d	What technical solution are used in order to ensure the wanted quality on the architecture?
Q5	Describe how the architecture is validated.
Q5a	How is it validated that the architecture meets the identified requirements?
Q6	Describe how the architecture is distributed.
Q6a	How is the architecture presented and stored?
Q6b	What architecture views are used? Why are they?
Q6c	How are the architecture views presented and stored?
Q7	Describe how it is ensured that software architecture has its self-evident place in the software development process.
Q7a	How do you ensure that the architecture rules and visions are used in the rest of the organization?
Q7b	How is the feedback on the architecture work collected
Q7c	How is it ensured that the product-line architecture is conceived as positive among the users of the architecture
Q8	How is the product-line architecture controlled?
Q8a	How is it ensured that the architecture has a positive impact on the organization.

Table 1. Questions in benchmarking instrumentation.

Id	Question
Q8b	How is ensured that the correct architecture vision is applied by the architecture group?
Q8c	How is the wanted architectural quality ensured when the architecture is evolved?
Q8d	How are the causes of product-line architecture erosion identified?
Q8e	How is it ensured that the product-line architecture is used in a correct way?

The benchmarking instrument that was used in the case study guides the participating companies to present their approach in the area. It would be possible to include questions that guide the companies to present other dimensions of their PPMPs as described in Section 2.1. For example, the results of this approach could be presented. However, in this case there was not enough data available. The results are instead captured in the review questions as described in Section 2.2.3.

2.2.2 Development of presentation

The presentation should preferably be developed by persons that are familiar with the organisation, but it can be developed in different ways. It may, for example, be developed by one person that already has knowledge of most of the work in the area, and therefore directly can describe the work. It may also be developed by one person that gathers information from the work in a structured way.

In the first case, a presentation can initially be developed and then be internally reviewed by appropriate persons in the organisation. In the latter case the person that is responsible for developing the presentation may first interview people in the organisation and then develop a presentation that can be internally reviewed by appropriate persons in the organisation.

In the case study, presented in Section 3, it was decided that the presentation should be written in natural language, since this was considered the most natural for the questions that were defined.

2.2.3 Review of presentation

There are two major objectives of the reviews. The first is that the organisations that have developed the presentations should receive feedback on their work. The second objective is that people in the reviewing organisations should get information from another organisation. Both of these objectives are premiered by letting many people review the presentations. However, for practical reasons, such as limited available effort, there will be a limited number of people that are able to review the presentations. It is important to note issues of confidentiality, which may also limit the number of people that review the presentations.

A likely procedure is that a number of people review each presentation and that one person is responsible for summarizing the findings and producing a report with feedback. After that, the feedback report is presented to the organisation that has produced the presentation in step 4 of the process (see Figure 1).

In order to guide the reviewers during the review and in order to obtain interesting feedback to the other organisation a review template with five review questions has been used. In the first two review questions the reviewer should assess the maturity of the presented approach and the result that it produces with a grade (0-10). The template includes a description of some of the grades. The intermediate are, of course, also allowed. The two quantitative review questions are presented in Table 2.

Table 2. Quantitative review questions.

Id	**Formulation**	**Description of grades**
RQ1	Evaluate the maturity of the presented approach and grade it (0-10)	0: no approach presented 1: An example of how it was once done is presented. 5: According to the presentation, most work is carried out in a methodological way. 10: A systematic procedure is used in every situation without exception. Systematic improvements are integrated in the approach.
RQ2	Evaluate and grade (0-10) the results that you believe that the presented approach gives to the presenting organisation.	0: There is no (or a negative) result of using the presented approach. 5: Most results are good. The approach seems to be a good way of working. 10: The best possible result is obtained. No other approach would give a better result.

For the three other review questions the reviewer should state the answers qualitatively. The two qualitative review questions are presented in Table 3.

All review questions (RQ1-RQ5) should be answered for every area in the description (according to the major questions in the benchmarking instrument, Q1-Q8).

Table 3. Qualitative review questions.

Id	Formulation
RQ3	Explain your reasons for the grades in questions RQ1 and RQ2.
RQ4	Present suggestions for improvements for the presenting organisation. It is important for the presenting company that you answer this question.
RQ5	Additional comments.

The intention is that the most important information transferred to the other company should be provided by the reviewers in RQ3 and RQ4. The objective of the first two review questions is to obtain data that allows comparison of the different reviewers and to let the reviewers reflect over the issues in the questions. This is why the order of the questions has been chosen in this way. If you are first asked to grade an approach, you will probably evaluate it carefully and compare it to the other approaches, etc.

3. Case study

The benchmarking approach has been evaluated in a case study [13, 14]. The research objectives of the study are to evaluate the importance of the areas covered by the questions in the benchmarking instrument, and to get experience from carrying out a benchmarking initiative according to the process presented above. The case study is carried out as a cooperation between two organisations:

- ABB Automation Technology Products AB: The organisation develops real time systems that operate in industrial environments. The systems have a life-cycle of about 10 to 30 years. Goals of the architecture are hardware and operating system independency, easy extension, openness, high reliability, small size, and high performance. Many products are built from the same system platform. The developed products are part of a larger system. The configuration of the used systems may differ greatly, from single components to large plants or geographically spread out configurations.
- Ericsson Mobile Communications AB: The company develops consumer products within the telecommunication area where competition has increased drastically during recent years. Consumer products for mobile Internet will be cheaper, smaller and more

complex and to be able to compete in this cutting-edge market, Ericsson Mobile Communication focuses on developing a software architecture that can easily be extended, maintained and reused. The purpose of the software architecture is to provide a common platform that can be used as a basis when developing consumer products with low cost and small effort.

Lund University and both of the industrial organisations are part of the same research network (LUCAS, Center for Applied Software Research[1]). It was therefore natural to involve the two industrial organisations as participants in the case study.

In the benchmarking approach a number of additional actions were taken in order to be able to draw conclusions. In order to receive feedback on the questions in the benchmarking instrument, a number of feedback questions (FQ) were formulated and distributed together with the review template to the two organisations. Everyone who reviewed the descriptions also replied to the feedback questions. This means that feedback information (FI) was produced by all participants in the study. The research actions and the usage of the research instrumentation are described in Figure 2.

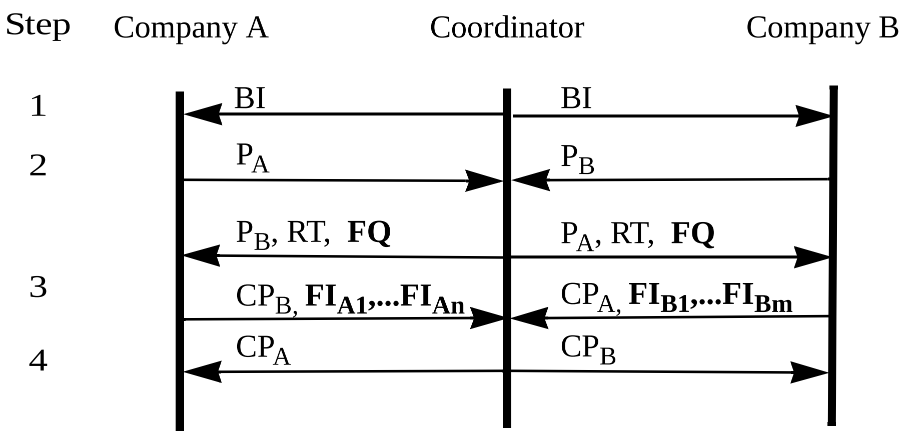

Figure 2. *The benchmarking process with research instrumentation (FQ - Feedback Questions, FI - Feedback Information) in bold writing. n denotes the number of reviewers at Company A and m denotes the number of reviewers at Company B.*

1. Further information on LUCAS can be found at http://www.lucas.lth.se.

Two feedback questions were formulated. Everyone that reviewed material from the other company answered the feedback questions for every general question in the benchmarking instrument. The answers to the feedback questions were given quantitatively with a grade from 0 to 10. The two feedback questions are presented in Table 4.

Table 4. Feedback questions.

Id	Formulation	Description of grades
FQ1	Grade (0-10) the importance for your own organisation of the issue that is covered by the question.	0: Not important at all. 10: Our most important question.
FQ2	Grade (0-10) the use of the answer to the question.	0: The answers to the question cannot help us at all when we compare to our own organization. 5: The presented approach can help us to find changes to the procedures at our company. Or, parts of the presented approaches may be directly introduced in our organisation. 10: The procedures that are presented in the answer are directly applicable in our organization and they would without doubt help us to improve with respect to quality, cost and lead time in our projects.

3.1 Analysis and results

In total, 11 persons answered the feedback questions. There were 6 persons from one of the organisations, organisation A, and 5 persons from the other organisation, organisation B[2]. The result of the analysis with respect to the feedback questions FQ1 and FQ2 are displayed in Figure 3 The result is displayed as mean values and error bars representing one standard deviation above the mean and one standard deviation below the mean. A small error bar shows that the reviewers' answers to the question were in close agreement.

The benchmarking instrument is analysed with respect to the major categories of questions, i.e., Q1, Q2,... Q8 in Table 1. This approach was chosen, because it was deemed too hard for the reviewers to distinguish

2. For confidentiality reasons it is not published which one of the involved organisations is denoted “Company A” and which organisation is denoted “Company B”.

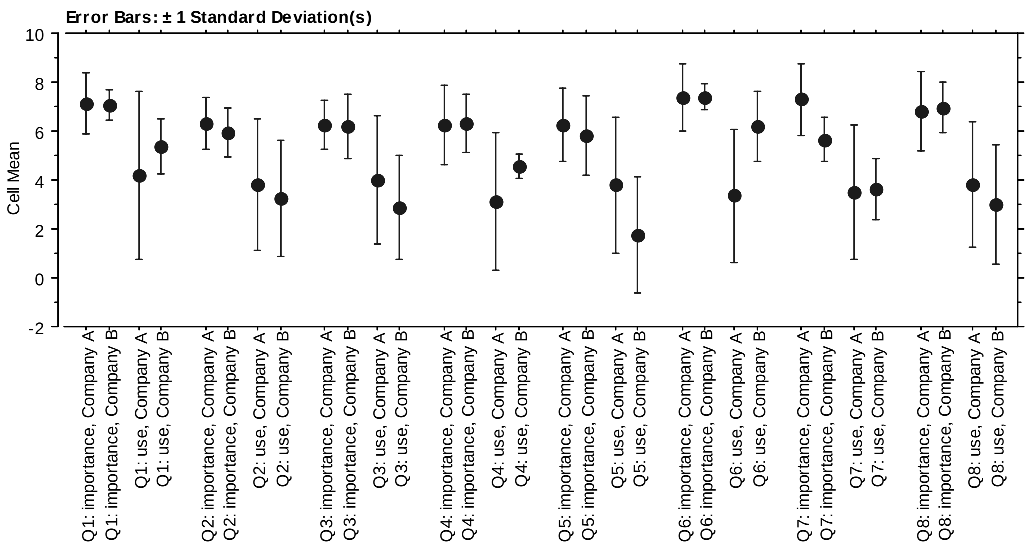

Figure 3. *Analysis-results for the feedback questions.*

between their opinions about the different sub-questions, Q1a, Q1b, etc. Most reviewers answered with respect to the major questions, but a few reviewers answered with respect to both the major questions and the sub-questions. For those persons a major score was estimated as the mean value of the major question and the sub-questions. For example, for the first major question the general answer would be calculated as (Q1+Q1a+Q1b+Q1c)/4.

The results in Figure 3 are shown for each major question (Q1-Q8), organisation and feedback question (FQ1, FQ2). FQ1 is denoted "importance" and FQ2 is denoted "use" in the figure. It can be seen that all questions are important for both organisations. No question seems to be less important than any other question. It can also be seen that the use of the descriptions of the other organisation is given a lower grade than the importance of the question by both organisations. In some questions organisation A gives a higher grade to FQ2 than organisation B, and in some cases it is the other way around. This may be interpreted as the organisations learn different things from each other in different areas. It is not only that the areas are considered important, it also indicates that it, in this case, was possible to use descriptions from the other organisation in order to identify improvement proposals. For example, the result for Q6 shows that both organisation A and organisation B have given a high rate to the importance of the question. This can be interpreted, as the area of work covered by the question is important for both companies. For the same question, the result shows that the companies have rated the question differently when considering the usage. Organisation A has given Q6 a low rating with respect to the usage of the answers to Q6 given by organisation B. Organisation B has on the other hand, given a high rating to the answers to Q6 from organisation A.

To summarise, it can be stated that the questions were appropriate for the organisations involved in this benchmarking initiative.

4. Conclusions

It can be concluded that the presented benchmarking approach was feasible in the case study. All major areas of the benchmarking instrument were regarded as important by both organisations. No areas in the benchmarking instrument was considered significantly less interesting than the other areas. This means that no major changes must be made to the instrument before it is further used and evaluated. Both organisations were interested in acquiring knowledge from the descriptions that were developed by the other organisation. It should, however, be noted that both organisations were involved in developing the instrument.

It is important to evaluate the validity of the results. In this case we believe that the most important issue to evaluate is the possibility to generalisation of the results. It is hard to estimate the possibilities of generalisation of the study. Further studies are needed in the area to be able to draw general conclusions. Another threat is that the organisations have been active in developing the instrument that later on has been evaluated in the case study. This also makes it hard to draw general conclusions, although we do not believe that it affects the other results to any large degree.

There are a number of areas that could be investigated in further work. Some time after the case study it would be possible to investigate what changes that have actually been introduced in the organisations as a result of the initiative.

It would also be interesting to carry out an evaluation as a self-assessment instead of involving two companies in a benchmark approach. Possible further research issues, of course, also include further case studies involving other companies.

Acknowledgments

This work was partly funded by The Swedish Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Lund University (LUCAS). The authors would also like to thank all the participants in the study.

References

[1] Bosch, J., '*Design & Use of Software Architectures: Adopting and Evolving a Product-line Approach*', ACM Press/Addison-Wesley, 2000

[2] Cohen, S., Gallagher, B., Fisher, M., Jones, L., Krut, R., Northorp, L., O´Brien, W., Smith, D., Soule, A., 'Third DoD Product Line Practice Workshop Report, Software Engineering Institute', Technical Report, CMU/SEI-2000-TR-024, 2000

[3] Narahari, Y., Viswandham, N., Kiran Kumar, K., 'Lead Time Modeling and Acceleration of Product Design and Development', IEEE Transactions on Robotics and Automation, 1999, Vol. 15, No. 5, pp. 882-896

[4] Tersine, R.J., Hummingbird, E.A.,, 'Lead-time Reduction: The Search for Competitive Advantage', International Journal of Operations & Production Management, 1995, Vol. 15, No. 2, pp. 8-18

[5] Wheelwright, S.C., Clark, K.B., '*Leading Product Development: The Senior Manager's Guide to Creating and Shaping the Enterprise*', The Free Press, New York, USA, 1995

[6] Humphrey, W., '*Managing the Software Process*', Addison-Wesley, 1989

[7] Bergman, B., Klefsjö, B., '*Quality from Customer Needs to Customer Satisfaction*', Studentlitteratur, Lund, Sweden, 1994

[8] Camp, R., '*Benchmarking - The Search for Industry Best Practices that Lead to Superior Performance*', ASQC Quality Press, Milwaukee, Wisconsin, USA, 1989

[9] Steepels, M.M., 'The Baldridge Award and ISO 9000 in the Quality Management Processes', *IEEE Communications Magazine*, 1994, October, pp. 52-56

[10] Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V., 'Capability Maturity Model for Software, Version 1.1', Software Engineering Institute, CMU/SEI-93-TR-24, 1993

[11] National Institute of Standards and Technology, 'Criteria for Performance Excellence, Baldrige National Quality Program', National Institute of Standards and Technology, USA, 2001. Available via http://www.quality.nist.gov

[12] Swedish Instiute for Quality, 'The SIQ Model for Performance Excellence', Swedish Instiute for Quality, SIQ, 2000. Available via http://www.siq.se

[13] Robson, C., 'Real World Research', Blackwell Publishers Ltd., Oxford, UK, 1993

[14] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., '*Experimentation in Software Engineering, an Introduction*', Kluwer Academic Publishers, 2000

Paper IV:

Tracking Degradation in Software Product Lines through Measurement of Design Rule Violations

Enrico Johansson and Martin Höst

Submitted to the *14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Ischia, Italy, July, 2002

Abstract

In order to increase reuse, a number of product versions may be developed based on the same software platform. The platform must, however, be managed and updated according to new requirements if it should be reusable in a series of releases. This means that the platform is constantly changed during its lifecycle, and changes can result in degradation of the platform. In this paper, a measurement approach is proposed as a means of tracking the degradation of a software platform and consequently in the product line. The tracking approach is evaluated in a case study where it is applied to a series of different releases of a product. The result of the case study indicates that the presented approach is feasible.

1. Introduction

Software organizations use tracking to promote efficiency in management and development of software. The tracking consists of monitoring and controlling that organizations stay within wanted cost, quality and lead-time boundaries. This is also valid for the tracking approach presented in this paper, though focused on software platforms and product line architectures [1][2][5][12][17]. The importance of tracking is accentuated in software platforms, the core asset of product line architecture. This is due to the fact that the same platform architecture supports a large number of products with guide-lines and mechanisms and is as such often in the critical line of product projects. The benefit of this kind of reuse is that the quality is maintained and lead-times improved when building many products on the same platform [15] [19].

Since the software platform is one of the core reusable assets of a product line architecture it has influence on the quality of the whole software product line. The quality is regarded as the totality of features and characteristics of the software platform that bears on its ability to satisfy stated or implied needs from the product line [10]. Therefore, tracking the deviation from a wanted platform structure (i.e. "erosion" [18] and "software aging" [16]) can communicate knowledge concerning the degradation of the whole product line. The degradation is characterized by a product line to pass from a higher grade to a lower grade of accomplishment of its purposes. This concerns the possibility to build different of versions and a family of products based on it.

The objective of the research presented in this paper is to identify and evaluate a measure for tracking degradation in product lines. The measure is based on design rules and a graph representation of the design rule violations.

The benefits of using measurement to improve and track the development and maintenance of software has been recognized in many studies, see e.g. [3]. Setting-up a measurement program for software platforms based on the measure presented is simple and can be highly automated. The measure has the possibility to detect the trend when platform erosion starts, and can cause the product line to degrade. Computing the presented measure at consecutive points in time gives the possibility to track the degradation over desired platforms versions or product families.
Similar studies have been made to investigate and present quantifiable metrics for degradation of software structures. Approaches to explicitly find architectural metrics have been presented in several studies, for example [4][11][14] however their main objective has not been the tracking of degradation of product lines and nor has graph measurements been used.

This paper is organized as follows. In Section 2, the usage of design rules in a product line is presented. The measure is defined in Section 3. The case study and the results derived from it are discussed in Section 4. In Section 5, conclusions and suggestions for further work are presented.

2. Design rules in platform development

"The structure of the system is given by a set of design rules, a structure can be seen as the architecture of the system" [7]. This statement is also true for the structure of a software platform. The structure of the system is however complemented by adding elements of implementation to the picture. The implementation elements can take the form of components, interfaces, connection mechanism or whatever units of high-level design that are used in the platform [8]. The implementation elements together with the design rules define the physical view of the architecture [9]. The way of building software products with these elements is described by a set of design rules, which the developers are required to follow when developing software products. The set of design rules communicates an architecture that is believed to be appropriate [13]. The usage of design rules can be summarized as follows:

1. A design rule describes the prescribed usage of the components, mechanisms and interfaces.
2. If a design rule is applicable to a component, mechanism, interfaces or a combination of these, the rule must be followed.
3. It can be the case that a design rule is not applicable for the product being developed. Some of the design rules may only be valid under some specific conditions. If these conditions are not met, the design rule is not applicable.

During the development of a product, several versions of design rules may be released. This can result in situations where a design that previously was considered as violations against the design rules is accepted and not considered as violations any more. The opposite can also be the case. Non-violations become violations because of new design rules that invalidate the design already in place. Therefore, the tracking approach suggested in this paper take into consideration both different releases of the platform and different releases of design rules.

The reasons for violating the design rules can be grouped into four categories, which relate to lead-time, quality and cost discussions of software development. The four reasons are defined by Perry and Wolf [18]: The first three concern architectural deviation and the forth concerns architectural drift.

Lead-time: Following the design rules will result in missing a market deadline and the economic revenue.

Quality: Following the design rules will result in missing the quality goal (e.g. throughput) of the software products, which are based on the platform.

Cost: The cost of following the design rules will cause the software project to miss its economical goal.

Knowledge: The developers using the design rules to develop a software product do not have the knowledge to use them correctly.

3. A measure of degradation

In this paper, a measure for tracking and quantifying product line degradation is presented. The tracking is performed by investigating the number of violations of design rules in a software platform. The measure can be used for tracking different releases and versions of a product, as well as the platform itself. However, before describing the measure and its proposed usage, some basic concepts concerning development, usage and evolution of software platform and software products are clarified.
In a product line development project the platform architecture is reused to develop several versions of a software product and even product families. Design rules are, as discussed earlier, a vital part of the architecture. During the development, these design rules may be removed, added or changed in order to preserve or change wanted quality attributes. These changes can be considered as violations to the original design rules that were defined to uphold a specific architecture in the platform. The violations will cause the product to deviate from the original perception of how an optimal platform and product architecture should look like. This deviation can give knowledge of a possible indication of degradation in the product line. This section describes the properties of a tracking approach based on a graph measure.

3.1 Graph measure properties

In a product line there can exist different versions of products. The product versions *1* to *s* are denoted as: P_1, P_2,..,P_s. In the same product line there exist different releases of approved design rules. The design rule releases *1* to *t* are denoted as: $DR_1, DR_2,..,DR_t$.

The graph measure can be seen as a function of P_i and DR_j for all *i* and *j*, $m(P_i,DR_j)$. This denotes the deviation of the architecture structure for product i compared to the wanted structure defined by release *j* of the design rules. The structure of P_i can be represented by a finite non-directed graph containing arcs and nodes. The nodes represent the implementation units described in Section 2. It is important that the arcs define the existence of a relationship between nodes, not the number of relationships. Nodes may be represented by points, and arcs are represented by a curve connecting the two points.

The arcs (relationships) that are not allowed by the design rules are considered as design rule violations. In the same way arcs only represent the existence of relationships. The arcs defining violations of design rules only depict the existence of violations, not the number of violations between nodes nor the type of the relationship between nodes.

A measure is meaningful and well formulated if the properties that are described below are true. The properties are based on the tree impurity measure properties presented by Fenton [6]. The four properties are as follows:

Property 1. $m(P_i,DR_j) = 0$ if and only if P_i does not violate any design rules of DR_j.

Property 2. $m(P_k,DR_j) > m(P_i,DR_j)$ if P_k differs from P_i only by the insertion of an extra arc (representing a violation of a design rule of DR_j)

Property 3. Let n_i denote the number of nodes in P_i and $violations(P_i,DR_j)$ the number of violations against design rules DR_j. Then let P_k be an increment of P_i by having added new nodes to the product according to the design rules DR_j. Both products use the same versions of design rules. Then the following is valid:

if $n_k > n_i$ and $violations(P_k,DR_j) = violations(P_i,DR_j)$

then

$m(P_k,DR_j) \leq m(P_i,DR_j)$

That is, the graph of P_k has more nodes than the graph of P_i, but in both cases, the arcs that represent violations against the design rules are the same. The graph impurity should be greater or equal for P_k compared to P_i. The property formalizes the intuitive notion that the measure should be smaller if the number of violations are the same but the system is larger.

Property 4. Let *violations*(P_i,DR_j) denote the number of violations of design rules DR_j in product P_i. The maximum value of $m(P_i,DR_j)$ for any DR_j and P_i is reached only when the product P_i violates all possible design rules possible for the product.

3.2 Definition of the graph measure

We can define a measure that satisfies all four properties: Let *number_of_violations*(P_i,DR_j) denote the number of violations against the design rules DR_j which are present in P_i. Let *max_number_of_violations*(P_i,DR_j) denote the maximal number of violations against the design rules DR_j which are possible in P_i. A measure can be defined as the following:

$$m(P_i,DR_j) = \frac{number\_of\_violations(P_i,DR_j)}{max\_number\_of\_violations(P_i,DR_j)} \qquad (1)$$

If the design rules DR_j allow the *max_number of violations*(P_i,DR_j) to be zero the measure is not valid. However, there is no sense in measuring violations against the design rules, if no violations are possible to make.

It is assumed that the number of violations can be measured by calculating the number of arcs in a finite non-directed graph. The graph represents the structure of the product, where the nodes represent architectural decomposition units and the arcs represents a dependency between the decomposition units. The number of arcs in P_i considered as violations against the design rules DR_j, are denoted as *arcs_violations*(P_i, DR_j). The nodes and arcs are also used to calculate the maximum number of possible violations against the design rules DR_j for a product P_i. The maximum number of arcs in a finite non-directed graph with n_i nodes is subtracted by the number of allowed arcs. The number of allowed arcs given by the design rules in release DR_j for product P_i is denoted $arcs\_allowed(P_i,DR_j)$. If n_i denotes the number of nodes in P_i, then the maximum number of arcs in a finite non-directed graph with n_i nodes, is equal to $n_i(n_i-1)/2$. Thus, equation (1) can be rewritten as:

$$m(P_i,DR_j) = \frac{arcs\_violations(P_i,DR_j)}{n_i(n_i-1)/2-arcs\_allowed(P_i,DR_j)} \qquad (2)$$

3.3 Properties of the proposed measure

Below it is shown that the four properties are true for the proposed measure.

Property 1. The consequence that P_i does not violate any design rules DR_j is that $arcs\_violations(P_i,DR_j) = 0$ and this makes the equation (1) equal to zero.

Property 2. If P_k differs from P_i only by an extra arc (representing a violation of a design rule of DR_j) the value of the denominator in equation (1) is equal for both $m(P_k,DR_j)$ and $m(P_i,DR_j)$. The relation $m(P_i,DR_j) > m(P_k,DR_j)$ can thus be simplified to $arcs\_violations(P_i,DR_j) > arcs\_violations(P_k,DR_j)$ which was a prerequisite for the property.

Property 3. For the property to be true, it must be shown that the maximum number of violations always are larger for P_k compared to P_i. P_k and P_i are different versions of the same product and the number nodes (decomposition units) are larger in P_k compared to P_i. Both versions of the product uses the same version of design rules. By adding new nodes more possibility to violating the design rules are given. Thus, the maximum number of violations is increased for P_k compared to P_i. This is the property that should be shown. The equality stated by property 3 is true when the number of violations are 0 for both products (i.e. property 1 is satisfied).

Property 4. For any given P_i and DR_j the value of $n_i(n_i\text{-}1)$ and *arcs_desginrule*(P_i,DR_j) is fixed. This means that the maximum value of equation (4) is given by the maximum of numerator *arcs_violations*(P_i,DR_j). The maximum value of *arcs_violations*(P_i, DR_j) is reached when it denotes all possible violations in product P_i using DR_j. The numerator *arcs_violations*($P_{i,}$,DR_j) would the be equal to the denominator *max_number_of_violations*(P_i,DR_j). The measure will then take the value 1.

4. Usage implications of the measure

Using the design rule violations as an integral part in a software measurement program puts strong requirements on the design rules. They should be handled and stored in a formal way. The design rules should preferably be stored in a version handling database, be categorized and have high visibility in the organization. By treating the design rules as any other artifact (e.g. requirements, test cases, source code) of the platform development will make this possible. This would be facilitated if the design rules had their own natural place in the version handling structure of the platform. The design rules can be documented in different categories of quality aspects (e.g. maintainability, performance, usability, etc.). This would enable the measurement of the degradation for these particular aspects of quality (e.g. degradation of maintainability, degradation of performance, degradation of usability, etc.).

Violations against the design rules can be found by examining the source code of the product and the platform. The source code can be regarded as the building block of the software, and it is up to the measurement method to extract the information of interest and to process it to show the aspect of interest.

The measure (1) can be used in the following three scenarios. Scenario 1 is concerned by a long-term aspect of degradation of the platform degradation while Scenario 2 and Scenario 3 are concerned with tracking short-term aspect of the degradation.

Scenario 1. This scenario describes the rationale of measuring platform degradation for product version i with respect to the first release of design rules, i.e. *m*(P_i,DR_j). The measure can be used to track how well the initial design meets the requirements from a product built on the platform. A great increase in the degradation measure can indicate that the intended design or process was improper and many modifications had to be performed in the platform during the development of the product. This knowledge can give the developers of the software platform relevant incentives for improvements when designing a new platform for the next product families. Project managers can use the knowledge to assess the risks and resources of a project to develop a new product family. A low degradation could mean that the platform could be reused, while a high value could mean that the complete platform and the development process supporting it must be redesigned. The redesign should take into consideration the causes of degradation and find remedies against them.

Scenario 2. This scenario describes the rationale of measuring platform degradation for product version i with respect to the last release of design rules, i.e. $m(P_i,DR_j)$. This measure can be used to keep of track of the platform's degradation in the current project development. The optimal outcome would be when that the degradation is zero. When the measure is plotted against different versions of the product, a horizontal line should visualize this. The measure can be of interest for the project managers and developers of the platform in order to track this type of degradation. The knowledge visualized by the tracking can be used by developers to find corrective actions before the degradation gets out of hand. Project managers can use the knowledge from the tracking, in an early stage, to detect potential quality and lead-time risks in the product release.

Scenario 3. This scenario describes the rationale of measuring platform degradation for product version i with respect to the previous release of design rules, i.e. $m(P_i,DR_{j-1})$. This measure can be used to track degradation in conjunction with the one described in scenario 2. The trend in degradation describes if the action taken to stop the degradation tracked by scenario 2 has been successful. If the comparison shows that the tracked degradation is still the same, the action done did not have any effect. In the worst case, it could be that the action taken have increased the degradation. The scenario described should primarily be of interest for the developers of the platform and the project managers of the product. This can be motivated with the same argument as in scenario 2.

5. The case study

The case study is conducted by collecting data from the product line and calculating the degradation measure proposed in Section 3. Finally, it is discussed whether the case study indicates that the measure can be used to track product line degradation.

5.1 Ericsson Mobile Communications AB, Sweden

The measurements are performed on a product developed by Ericsson Mobile Communications AB. Ericsson Mobile Communications AB used a product line approach for developing consumer products within the telecommunication area. The purpose of the platform in the product line is to provide common software that can be adapted to a variety of consumer products with low costs and small effort. The platform is an embedded system with support for a variety of functionality, for example wireless protocols, data communication protocols and multimedia services.

The software architecture of the studied software product is basically a client-server solution, where software components provide public interfaces that are to be used by other software components. Standardized architectural mechanisms for communication between software components are defined and used throughout the system. The aim is to have a set of stable software components and mechanisms that can be reused between products without major redesign or rework.

5.2 Detecting design rule violations

It is important for the measure that the structure (i.e. architecture) symbolizes some sort of relationship between the components. These relationships can be in the form of source structure or behavior structure [20]. The source structure involves static dependencies while the behavior structure involves dynamic interaction dependencies.

The case study investigates the behavioral structure, as violations against design rules for intra-module invocations. A module is a high level component of the system. Design rules defining the usage of function calls and intra-module calls are therefore possible to check with this method. Design rules stating which components can be called by another component can be modelled in a graph. The arcs denote the invocations between components and the nodes denote the components themselves. Therefore, a graph showing the intra-module components invocation is feasible to use as input for the measure proposed.

5.3 Performing the measurement on a selected product

The measurement is based on data collected from five different development releases of the product stored in a configuration management (CM) database. It has been possible to retrieve all the artifacts used and generated by the building process of the product. The source code, make-files and other artifacts have been used to check for design rule violations in the product. Five releases were selected to cover the life cycle of the product stored in the database in a uniform way. This is achieved by having chosen the same number of stored releases between each of the measurement points.

All the five releases of the source code are checked in order to find violations against the design rules of Scenario 1, mentioned in Section 4.

The result from the measurements of the five releases is presented in Figure 1.

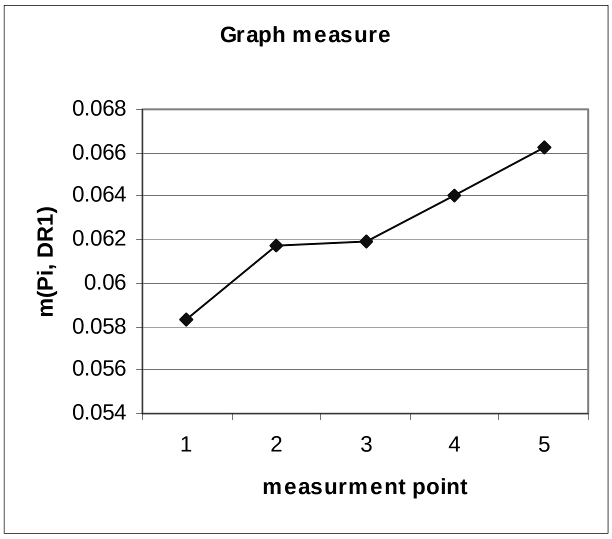

Figure 1. *The degradation measure is plotted against five measurement points (i.e. five development versions of the product).*

5.4 Result from case study

The curve in Figure 1 shows the measure per measurement points. Considering the curve, it can be noted that the gradient of the curve is positive not negative. It can also be noted that the gradient is not constant. In other words, Figure 1 can be interpreted as the product line is degrading with non-constant rate between product versions. This interpretation was confirmed by the organization in charge of the platform where the case study has been performed. They confirmed that the trend for the product is that the deviation from the wanted structure generally increases for every version of a product based on the platform. This shows that using the gradient of the proposed graph measure gives the possibility to track degradation in the product line.

Despite this encouraging response of the case study, a set of issues that still must be overcome to empirically validate the measure. The issues are the following:

- Other variables from the platform also have an increasing trend and could therefore be used to track the degradation. For example, the number of design rule violation also shows an increasing trend. It can be argued that only studying the violations would be a good enough tracking measure. This is true, but one of the prerequisites of the measure was that it should be normalized with the size of the platform. This is not true for a measure containing only the number of design rule violations. There is, however, a need for a comparative study of different possible measures (including the graph measure). The study should quantitatively compare the trends of degradation to the trends in the measures.
- The rise of the graph measure calculated in the case study is very small, in fact so small that the rising slope can be considered as noise. To really investigate this issue larger case studies should be performed and a larger number of design rules should be taken into consideration.
- The case study has only considered design rules that are related to the invocations of decomposition units. A complete set of design rules should be taken into consideration in order to have an opinion of the complete platform.

- The case study consists of measurement from one company and from only one product. Also, data is only collected on five measurement points. Case studies should be performed in other companies, more products and with a larger set of data.
- Only scenario 1 (see Section 4) is comprised in the case study. However, it is believed that the practical implication of this scenario can be related to the other scenarios.

6. Conclusion

A significant challenge for software product lines is to develop a software platform that maintains the wanted quality between products and product families. This article takes a step forward towards that goal by providing a method to track an eventual degradation of product lines. The contributions of the method are the following:

- A metric is identified that measures the deviation of the product from the original structure depicted by the design rules of the software platform. The measure is based on graph representation and design rule violations.
- The measure is evaluated in a limited case study. A goal of the method is that it should be practical and feasible to use in a company using the concept of software platforms. This is shown in the case study. More extensive, empirical studies must however be performed in order to validate the measures further.
- Different scenarios of how to use the measure are presented. The different scenarios describe the usage of the measures to track both short-term and long-term aspects of degradation. The short-term aspect is concerned with the degradation of the product line in the product being developed. The long-term aspect is concerned with degradation of software platforms between product lines. In all scenarios, stakeholder, usage and practical consideration are described.

In developing the basis for the measurement a general discussion of how to use design rules in a software platform environment are presented. Four properties of a graph measure are discussed and it is shown that they are applicable to the identified measure. The software platform process is described in light of the usage of platform design rules. Possible causes for degradation are described and it is discussed how they affect the usage and definitions of design rules. The design rules are an important property of many architectural descriptions. In addition, they open a way to analyze the system. In this case, the analysis is done by counting the number of violations against the design rules and relating them to the possible number of violations.

Acknowledgements

This work was partly funded by The Swedish Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Fund University (LUCAS). We thank employees at Ericsson Mobile Platforms AB, Sweden, particularly Jan Lind and Peter Lerup, as well as Fredrik Nilsson at Enea Realtime System, Sweden. A thanks also to Daniel Karlström at the Department for Communications Systems, Lund University, Sweden, for reviewing the article.

References

[1] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison-Wesley, 1998.

[2] Bosch, J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.

[3] Briand, L. C., Morasca, S., Basili, V.R., "*Measuring and assessing maintainability at the end of high level design*", Proceedings of International Conference on Software Maintenance, pp. 88-87, 1993.

[4] Carriere, S. J., Kazman, R., Woods, S.G., *"Assessing and maintaining architectural quality*", Proceedings of the 3rd European Conference on Software Maintenance and Reengineering, pp. 22-30, 1999.

[5] Clements, P., Northrop, L.M., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[6] Fenton, N.E., Pfleeger, S.L., *Software Metrics: A Rigorous and Practical Approach*, Revised. Boston: PWS, 1997.

[7] Gacek, C., Abd-Allah, A., Clark, B., Boehm, B., "*Focused Workshop on Software Architectures: Issue Paper*", USC Center for Software Engineering, 1994.

[8] Garland, D., Shaw, M., *An Introduction to Software Architecture*, Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific Publishing Company, 1993.

[9] Hofmeister, C., Nord, R., Soni, D., *Applied Software Architecture*, Addison-Wesley Longman, 2000.

[10] ISO 9000:2000 - Quality Management Systems-Fundamentals & Vocabulary

[11] Jaktman, C.B., Leaney, j., Liu, M.,"*Structural Analysis of the Software Architecture-A Maintenance Assessment Case Study*", Proceedings of 1st Working IFIP Conference on Software Architecture, 1999

[12] Jazayeri, M., Ran, A., Van Der Linden, F., Van Der Linden, P., *"Software Architecture for Product Families: Principles and Practice," Addison-Wesley,* 2000.

[13] Jones, A.K., *"The Maturing of Software Architecture"*, Software Engineering Symposium, Software Engineering Institute, Pittsburgh, PA, 1993.

[14] Kazman, R., *"Assessing architectural complexity",* Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering, pp. 104-112, 1998.

[15] Meyer, M., Lehner, A.,"*The Power of Product Platforms*", The Free Press, New York, 1997.

[16] Parnas, D.L.,"*Software aging*", Proceedings of 16th International Conference on Software Engineering, pp. 279-287, 1994.

[17] Parnas, D.L.,"*On the Design and Development of Program Families*", IEEE Transaction Software Engineering (special issue), SE-2, 1976.

[18] Perry, D.E., Wolf, A.L.,"*Foundations for the Study of Software Architecture*", Proceedings of ACM SIGSOFT, Vol.17, No.4, pp.40-52, 1992.

[19] Sanderson S., Uzumeri, M.,"*Managing Product Families: The Case of the Sony Walkman*", Research Policy, No. 24, pp. 761-782, 1995.

[20] Stafford, J.A., Wolf, A.L., *"Architectural-level Dependence Analysis in Support of Software Maintenance*", Proceedings of 3rd International Software Architecture Workshop, pp.129-132, 1998.