TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Jakub Rudzki

# Software Quality Concerns in a Commercial Setting

Julkaisu 954 • Publication 954

Tampere 2011

Tampereen teknillinen yliopisto. Julkaisu 954
Tampere University of Technology. Publication 954

Jakub Rudzki

# Software Quality Concerns in a Commercial Setting

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB223, at Tampere University of Technology, on the 25th of February 2011, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2011

ISBN 978-952-15-2531-5 (printed)
ISBN 978-952-15-2539-1 (PDF)
ISSN 1459-2045

# Abstract

Quality improvements in small and medium-sized software organisations can be done in many ways. Companies can try to implement one of the well-established quality improvement frameworks. However, many reports from practitioners show that small and medium-sized organisations may not be prepared for fully-fledged and possibly expensive improvement programmes. An alternative to official quality improvement programmes may be an in-house initiative. Due to their size and often to the organisational culture small and medium-sized software organisations may encourage grass-root quality improvement initiatives.

In this thesis a grass-root initiative for a software quality improvement is presented. The initiative proposes an alternative path of quality improvement for small and medium-sized organisations utilising their size and culture. This thesis proposes five areas of quality improvements and concrete measures that can be taken for quality improvements in those areas. The improvement areas were selected based on the experiences gained while working in the case software company Solita. The areas were selected as reaction actions to observed needs for improvements, or as proactive actions preparing for known organisational changes.

The five improvement areas discussed in this thesis address selected improvement needs at the organisational level and those directly related to software development. The improvement needs derive from experiences in the case organisation, which uses subcontracting partners for development. The use of subcontractors also resulted in some projects being conducted as multi-site development. Therefore, the proposed improvements reflect the organisational settings. Concrete improvements are proposed for the outsourcing-partner selection process. Moreover, recommendations on agile methods usage in development projects with special emphasis on subcontractor cooperation are proposed. Furthermore, specific practices for multi-site development are proposed, as well as a framework for open source component selection for reuse in commercial software solutions. Finally, specific improvements in design of distributed systems are proposed.

Each of the improvement areas consists of concrete improvements that proved to be successful to various degrees. However, the variety of areas shows how a software organisation can practically implement an in-house software quality improvement initiative. In this thesis three research questions are answered. The questions focus on how concrete improvements can be implemented in selected areas of a software organisation's operations. Moreover, the questions aim at answering how an in-house quality improvement initiative can be implemented, and how it can be regarded as complementary to official improvement frameworks, which allows a soft-

ware organisation to implement a fully-fledged quality improvement programme when it is ready.

The proposed improvements in various areas were developed and evaluated in a commercial context of a software service company. Some of the initiatives resulted in changes in organisation processes, which indicated their usefulness. Generally, the presented experiences from implementing a grass-root level software quality improvement initiative show that such initiatives driven by individuals or groups of individuals within an organisation are possible. Therefore, other organisations in a similar environment may utilise the results presented as concrete solutions for concrete problems or as an inspiration for encouraging similar improvement initiatives in their organisations.

# Preface

I have been working on this thesis since autumn 2003. This project was planned to take only a few years, however, in the course of years it proved to be more time-consuming than I had anticipated. Despite the delay it was a very enriching experience. There are many people who helped me during the years in this work and now I would like to express my gratitude to them.

My thesis would not be possible without the enormous support and encouragement I have received from my supervisor, Professor Tarja Systä. I would like thank her for always providing me with great guidance and spreading very positive attitude that was particularly useful when I needed an energy boost. Additionally, I would like to thank other members of Software Systems Department at TUT for their support. Particularly, I would like to thank Professor Tommi Mikkonen and Dr Imed Hammouda, who in addition to research support co-authored papers with me, and Professor Kai Koskimies for his general guidance and support. Naturally, I would also like to thank the reviewers of this thesis Professor Casper Lassenius and Dr Andrea Capiluppi for their valuable comments and feedback.

Moreover, I would like to thank my other co-authors Kimmo Kiviluoma, Tuomas Mikkola, Karri Mustonen, and Tero Poikonen for contributing in my research. Additionally, I would like to thank other colleagues from Solita for their comments and input. Then I would like to thank Solita's management for allowing me to use company's data for case studies and financing some of the conference and training expenses, particularly I would like to thank Heikki Halme, Raimo Arvola, Jari Niska, Marko Torpo, Jukka Jääheimo, and Toni Uimonen. Additionally, I would like to thank Solita's IT Department, particularly Kari Hippolin and Antti Varjonen, for their technical support with equipment needed at the beginning of my research.

Furthermore, I would like to thank organisations that supported my thesis: Tampere Doctoral Programme in Information Science and Engineering (TISE), Graduate School on Software Systems and Engineering (SoSE), Ulla Tuominen Foundation, Academy of Finland, and Solita Oy.

Additionally, I would like to thank my friends, close ones, and family for their support and understanding when I was occupied with this thesis (naturally, the list could be much longer but you know who I mean): Eveliina and Mikko Nurmi, Sylwia and Paweł Rogowicz, Jakub Borkowski, Jari Asp, Asta Suokas, Oleg Kozitsyn, Lasse Palkivaara, Maija and Kimmo Lahtinen, Hanna Toimi and Reijo Anttila, Else and Jarmo Lehto, my late aunt Łucja Biernacka, my sister Barbara Miernik, Barbara and Grzegorz Łukawski, Riikka and Mikko Laakso, and surely Jan Malmström. Moreover special thanks for my friends who also helped me with proofreading of

my papers are owed to Karen Thorburn and Michelle and Sing Wee.

Finally and last but definitely not least, I would like to express my gratitude to my parents, Edward Rudzki and Magdalena Rudzka, for their unconditional love and support. I would like to dedicate this whole thesis to them, which I express in Polish:

*Dedykuję tą pracę moim rodzicom, Magdalenie i Edwardowi Rudzkim,*
*którzy okazywali mi bezwarunkową miłość i wsparcie.*

*November 2010, Tampere, Finland*

*Jakub Rudzki*

# Table of Contents

# List of Included Publications

This dissertation is a compilation of the following publications.

[P1] Jakub Rudzki, Imed Hammouda, Tuomas Mikkola, Karri Mustonen, and Tarja Systä. Considering Subcontractors in Distributed Scrum Teams. A chapter in book *Darja Šmite, Nils Brede Moe, Pär J. Ågerfalk, (Eds.) 'Agility Across Time and Space: Implementing Agile Methods in Global Software Projects'*, p. 235-255, May 2010, Springer 2010.

[P2] Jakub Rudzki, Imed Hammouda, and Tuomas Mikkola. Agile Experiences in a Software Service Company. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009*, pp.224-228, 27-29 Aug. 2009, IEEE Computer Society.

[P3] Jakub Rudzki, Kimmo Kiviluoma, Tero Poikonen, and Imed Hammouda. Evaluating Quality of Open Source Components for Reuse-Intensive Commercial Solutions. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009*, pp.11-19, 27-29 Aug. 2009, IEEE Computer Society.

[P4] Jakub Rudzki, Tarja Systä, and Karri Mustonen. Subcontracting Processes in Software Service Organisations - An Experience Report. In *Proceedings of the International Conference on Software Process, ICSP 2009*, pp. 224-235, Springer-Verlag.

[P5] Jakub Rudzki, Imed Hammouda, and Tommi Mikkonen. Ensuring Architecture Conventions in Multi-site Development. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008*, pp.339-346, 2008.

[P6] Jakub Rudzki and Tarja Systä. Performance Implications of Design Pattern Usage in Distributed Applications: Case Studies in J2EE and .NET. In *Proceedings of the ISSTA 2006 workshop on Role of Software Architecture for Testing and Analysis, ROSATEA 2006*, pp. 1-11, ACM.

[P7] Jakub Rudzki, Imed Hammouda, and Tommi Mikkonen. Tool Support for Quality-driven Design. In *Proceedings of the 3rd Nordic Workshop on UML and Software Modeling, NWUML 2005*, pp. 193-207, 2005.
`http://www.cs.uta.fi/reports/pdf/A-2005-3.pdf`

[P8] Jakub Rudzki. How Design Patterns Affect Application Performance - a Case of a Multi-tier J2EE Application. In *Proceedings of the 4th International Workshop on Scientific Engineering of Distributed Java Applications, FIDJI 2004*, pp. 12-23, Springer.

# CHAPTER 1

# Introduction

Software companies operate in a very competitive and dynamic environment of constant technological changes. They strive to provide the best possible products or services for their customers. Irrespective of the companies' business focus, the quality of products and services can be one of the main factors distinguishing them from competitors. Therefore, on one hand, software companies pay much attention to initiatives that may improve the overall quality in their organisations. There are many existing frameworks and initiatives that aim at improving quality of software, for example, Software Process Improvement related initiatives [25, 31, 100]. On the other hand, companies seek solutions that do not necessarily require much investment and too many changes in the existing company processes.

In this thesis the problem of software quality improvements is discussed in the context of a medium-sized [1] software service company. The company has not adopted any official quality improvement frameworks, yet since 1996 it has been successfully developing software. This thesis proposes selected quality improvement practices in different areas of company operations. The set of improvements is a grass-root level initiative for quality improvement in an organisation. The improvement areas include two categories relating to organisational practices and software development. Hence, the quality improvement practices in various areas are viewed from different perspectives representing different points of views on the software development. The categories consist of five improvement areas. The details of the areas and research objectives are presented in Section 1.2 after introducing general quality-related concepts in Section 1.1. Then an overview of contributions of this thesis in the various areas of quality improvement are presented in Section 1.3. Finally, the context of this dissertation is outlined in Section 1.4, and the organisation of the whole dissertation is presented in Section 1.5.

## 1.1 Software Quality Improvement Approaches

Let us start by reviewing some main concepts related to quality, which is the main theme of this thesis. Software quality is a complex subject as software is complex by its nature. In addition to the complexity of software itself, there are additional external and internal factors that influence software quality. Internal factors may

[1] A medium-sized company according to the European Commission has under 250 employees. http://ec.europa.eu/enterprise/policies/sme/facts-figures-analysis/sme-definition/index_en.htm

include the processes used for software development, tools, technologies, and personnel. External factors may include customer expectations, schedules, and cooperation with other parties contributing to the software solution being developed. All of these factors can affect the quality of software offered to customers. However, before addressing particular problems related to software quality the term itself must be defined.

Quality in general is seen differently by various parties. A variety of approaches to defining product quality have been presented by David Garvin [48], who lists five different approaches to defining quality:

1 *Transcendent*, which is a philosophical view on quality as something that can be recognised, but cannot be defined.

2 *Product-based*, which aims at determining quality by measuring specific attributes of a product.

3 *User-based*, which reflects the quality perceived by the user as satisfaction of user needs and expectations by a product.

4 *Manufacturing-based*, which sees product quality as compliance with requirements.

5 *Value-based*, which determines product quality though the cost and price of a product.

These approaches to defining quality show the differences in how quality can be seen by different stakeholders. These different points of view on quality have been used also for software quality. Kitchenham and Pfleeger [73] discuss software quality and its different meanings to different stakeholders. According to the IEEE's definition '*Software quality is the degree to which software possesses a desired combination of quality attributes.*' [115, p. 3]. Examples of quality attributes include performance, dependability, security, and safety [11].

A number of frameworks aiming at improving processes in IT companies have been created. The frameworks address, among others, different aspects of IT company operations. It is worth briefly going through these frameworks as they are referred to in the later parts of this thesis. An overview of different process improvement frameworks has been presented, for example, by Cater-Steel *et al.* [25], who list four popular frameworks: CMMI, ISO 9000, ITIL, and CobiT. Currently, the popularity of the frameworks being used is limited. However, we should first briefly categorise the frameworks listed by Cater-Steel *et al.* [25].

One of the well known frameworks aiming at improving the software development process is the Capability Maturity Model (CMM) [116], which was later extended to CMM Integration (CMMI) [117]. The models were developed by the Software Engineering Institute (SEI) of Carnagie Mellon University. The CMMI defines five levels of organisation maturity. The levels are initial, managed, defined, quantitatively managed, and optimising, listing from the lowest to the highest level. Each of the levels corresponds to specific practices that an organisation is supposed to use in order to fulfil requirements of the level. The CMMI model has also been refined for

product and services development as the CMMI Development model [124], which specifies 22 process areas relevant for development.

Another framework used for quality assurance is the ISO 9000 series standards supported by the International Organisation for Standardization (ISO). The ISO 9000 series standard includes specific parts on requirements in ISO 9001:2000 [40], fundamentals of quality management in ISO 9000:2005 [41], and guidelines in ISO 9004:2009 [42]. The ISO 9000 is a very generic quality assurance framework suitable for any organisation, naturally including software development companies. Moreover, ISO specifies a standard for information technology organisation processes maturity assessment ISO/IEC 15504 [39].

Furthermore, the Information Technology Infrastructure Library (ITIL) is another example of a quality assurance framework. ITIL has been developed by the UK's Central Computer and Telecommunications Agency (CCTA) [25, p. 3], but the new versions of ITIL are managed by the Office of Government Commerce (OGC) [95]. The ITIL framework is meant as a framework for IT service providers. Finally, the CobiT [62] framework, developed by the IT Governance Institute (ITGI), aims at IT governance. Neither the ITIL nor the CobiT are directly relevant to software development, but they are part of IT-related process improvements frameworks and as such are relevant for process improvement initiatives.

Software quality has been addressed by different Software Process Improvement (SPI) initiatives. An extensive literature review on SPIs in small and medium-sized software companies has been presented by Pino *et al.* [100]. Their work examined a number of other publications in search for the most commonly mentioned SPI method in the literature. The findings pointed out the SEI's CMM to be the first and ISO's 15504 and 9001 models to be the second most often reported models being used in small and medium-sized software companies. However, the work also reports that only two medium-sized companies out of 122 have obtained a formal certification for the model used. Additionally, this work points out reasons why SPI models are not used in SME companies. The main such reasons discussed include concerns that the standards are costly to implement in terms of time and money (Saiedian and Carr 1997) [110].

Other reasons why small organisations often do not use formal SPI models have been reported by Staples *et al.* [120]. They have investigated reasons why software companies in Australia do not adopt SPI programmes. Small software organisations generally justified their rejection of SPI (CMMI in that case) as being too small organisations for adopting such a programme. Other common reasons for rejecting SPI programme by companies reported in that research were: the programme being too costly, having no time, and already using another form of SPI [120]. The Australian study was later replicated by Khurshid *et al.* [72] in the context of the Malaysian software market. Their findings also showed similar results, at the general level, as the Australian study. The reported reasons of not adopting SPI programmes in the Malaysian study were: being too costly, no clear benefits, being a small organisation, and CMMI not being an organisational priority [72, p. 42]. Both of these related pieces of research showed that small software companies often do not adopt SPI programmes because they regard them costly and unsuitable for them.

Other aspects of introduction of SPI in companies of different sizes have been

presented by Dybå [31]. The research investigated 120 software organisations in Norway. The presented results show that small organisations can implement successfully some elements of SPI, but they do not necessarily implement fully-fledged SPI programmes. The conclusion of the differences between implementation of SPI in small and large companies was that the size of the organisation does not limit its potential for SPI success [31, p. 154]. Additionally, the study reported on differences between a large and small successful software organisation. The findings showed that large successful companies rely on formal procedures, while small successful organisations emphasise exploration of new possibilities and utilisation of creativity and diversity of their people [31, p. 154].

Another aspect of SPI in the context of SME was presented by Lester *et al.* [79], who investigated the relation between SPI and the growth of companies. The investigated companies did not officially have CMMI implemented, but the findings of the research showed that medium-sized companies tend to adhere to the CMMI practices more than small companies. The study showed which process areas of CMMI are most relevant to small and medium-sized companies. However, the main conclusion of the study was that CMMI's principles support growth of companies, and therefore SMEs should focus their efforts on the identified areas for their future growth.

A comprehensive review of SPI models for small and medium organisations have been presented by Deepti Mishra and Alok Mishra [87, 88]. The review compared characteristics of selected SPI models and additionally listed factors that small and medium organisations should take into account while selecting an SPI [87, p. 281] [88, p. 122]. The selected factors include, for example [87, p.281]:

- the fact that an SPI is based on an established model, e.g., CMM, which can enable the organisation to change the SPI easier in the future to the well-established one,
- the fact the an SPI takes into consideration specific needs of an organisation, and
- the fact that an SPI involves participation of the software development team members, which is perceived as a factor securing their confidence and commitment to the SPI.

These factors are especially important in the context of this thesis, as the proposed grass-root level quality improvements are specific to a particular context of an organisation, which addresses specific needs. Furthermore, the grass-root improvements by their nature involve organisation members from the improvement inception to its realisation. However, the extent of the involvement varied.

Benefits of implementing SPI in small companies have been also reported by Karlheinz Kautz [68]. Moreover that study recognised four success factors, namely, flexible approach to the improvement, network of small enterprises being involved in the improvement, which provides support for achieving a common goal, and external technical help and financial support for the improvement programme [68]. Furthermore, Kautz *et al.* [69] have formulated three pieces of advice for organ-

isations planning to introduce an improvement initiative. The recommendations are [69, p. 631]:

- use a structured model for process organisation,
- adjust the model to the specific context of the organisation, and
- perform the improvement activities as projects with defined roles, documents, responsibilities and roles.

Out of all the success factors and recommendations by Kautz *et al.*, in the context of this thesis the aspect of adjusting the improvement initiative to the organisational context is the common one. The examples of quality improvements presented in this thesis reflect the context of the organisation they have been conducted in. Finally, what is also specific to the presented improvements is the role of the change agent, who was directly involved in the improvements. This aspect of the SPI programmes was also recognised by Kautz *et al.* [70], who classified four types of process agents and two roles that they can play. The distinctive roles were of observers, who are not directly involved in the changes, and actors, who participate in the change [70, p. 17]. In this light the role of the author of this thesis can be perceived as an actor who has participated in the presented improvements.

Some tools are available to help companies to assess their processes with respect to the conformance to specific standards. One example of such a tool is MARES [131]. The tool was used by Gresse von Wangenheim *et al.* [130] for process assessment of compliance with the ISO/15504 [39] standard in small software companies. The assessed processes included: Supply, Requirement elicitation, Project Management, Software test, Measurement, Software construction, Customer support, Supplier tendering, Contract agreement, Software release, Software acceptance support, Software installation, Software integration, and Change request management [130, p. 95]. The long list of different processes assessed in 8 different software organisations shows how many different processes can be improved in different companies. The results of the study showed that also small companies can benefit from process assessment. All of the assessed companies found the results assessment beneficial and depending on their needs the companies started establishing processes most crucial for them [130, p. 96]. This study shows an application of a particular assessment method but also the needs demonstrated by even very small companies.

## 1.2 Research Objectives

As was presented in the previous section, the concept of quality can be seen from different perspectives, and there are many approaches to improving or assuring quality in software companies. In this thesis quality is viewed from the end user point of view. In the context of a software service provider company, the end user is the other organisation ordering a customised software solution. Therefore, the quality improvements in this context aim at satisfying directly or indirectly the expectations of the software solution orderer. Additionally, quality is seen through the value and

cost perspective of the software solution developed. However, even in this case the end customer assesses the value of the solution.

The main objective of this dissertation is to present how a small or medium-sized software organisation can implement an in-house quality improvement initiative by flexibly and in a light-weight fashion addressing its needs in selected quality improvement areas. It should be noted that in the context of this thesis the term *small, flexible and light-weight quality improvement* means an improvement initiative that is limited to a specific area of an organisation's operations, hence small. The flexibility of the approach derives from the fact that the implementing organisation can flexibly choose what should be improved. Finally, the process is considered light-weight when it does not require a significant effort in the organisation. In practice it also means that the process is not meant for any official appraisal. The improvement areas are selected reactively, when an existing problem or room for improvement is observed, or proactively, when an organisation is preparing for some changes and anticipates that new practices are needed for the changes. Furthermore, in a small or medium-sized software company such improvement areas can be suggested and tackled by individuals or small groups of individuals. Naturally, company culture must encourage such grass-root level initiatives for quality improvements. An in-house quality improvement initiative may be a path that is more suitable for small and medium-sized software companies than fully-fledged SPI programmes. Furthermore, a choice of a flexible and light-weight approach to quality improvements does not necessarily mean that the official SPI programmes cannot be followed in the future.

Therefore, this dissertation aims at answering the following research questions (RQ):

RQ1 *How can small quality improvements in different areas of a software company's operation help to improve software solution quality?*

RQ2 *How can a small or medium-sized software service company implement a flexible and light-weight quality improvement initiative based on its internal organisational experience?*

RQ3 *How can the flexible and light-weight approach to quality improvement complement an official SPI framework?*

The main research question, RQ1, will be evaluated in this dissertation based on the presentation of the different improvement areas and providing concrete examples of improvement actions. The question RQ2 will be evaluated based on quality improvements presented as answers to RQ1 and then regarded as a set of possible solutions that can be used at an organisation level for a quality improvement initiative. Finally, the question RQ3 will be evaluated based on comparison of the proposed quality improvements with process areas specified in a mainstream quality assurance framework, namely, CMMI for Development [124]. This dissertation's initial objectives have been presented at COMPSAC 2008 Doctoral Symposium [107].

This dissertation proposes local quality improvements in five different areas of activities in a software service organisation as measures for an overall quality improvement initiative. The areas selection and improvements proposed are derived

from experiences in the case company. Additionally, in many areas issues related to multi-site development, subcontracting, and agile practices are especially highlighted, as they reflect organisational changes in the case organisation. The improvement areas are logically grouped in two categories: *Organisation* and *Development*. These two categories cover two important aspects of a software organisation functioning. The *Organisation* category focuses on activities that are vital for software development, but that can be seen as ways of organising work. The *Development* category focuses on activities related to software development, which is the core activity in a software organisation. The categorisation is only a logical way of grouping the improvement areas, and some areas may have certain aspects that can be overlapping with both categories.

In this thesis the two categories consist of improvement areas as follows:

- *Organisation*
  - *Outsourcing Supplier Selection*, which determines the suitability of specialists coming from outsourcing partners for software development in SSP. The choice of specialists developing a software solution can directly affect the quality of software produced.
  - *Agile Project Practices*, which specify agile practices used for project organisation. The usage of practices that are suitable for a particular development organisation, e.g., encouraging communication and customer participation regardless of their location, affects the software quality and customer satisfaction.
- *Development*
  - *Multi-Site Practices*, which gather practices that are used in multi-site software development to solve particular problems. Development practices that are tailored for a specific development environment, i.e., a multi-site environment, help to improve the quality of the software produced regardless of the development arrangements, so that the end customer is not affected negatively by development arrangements.
  - *Component Evaluation*, which affects software system quality by showing how to evaluate components to be used in a software solution in a way that assures that only suitable components are accepted for use.
  - *Selected Design Solutions*, which improve software system quality by proposing specific design solutions and tool support that affect specific quality attributes. Improvements in specific quality attributes, e.g., performance, are directly visible to the end customer using the software.

Each of the improvements in these selected areas affects directly or indirectly the quality of software produced. Naturally, these quality improvements do not constitute all possible areas of improvement. This particular set of areas derives from experience gained in the case organisation. The context organisation is further discussed in Section 1.4.

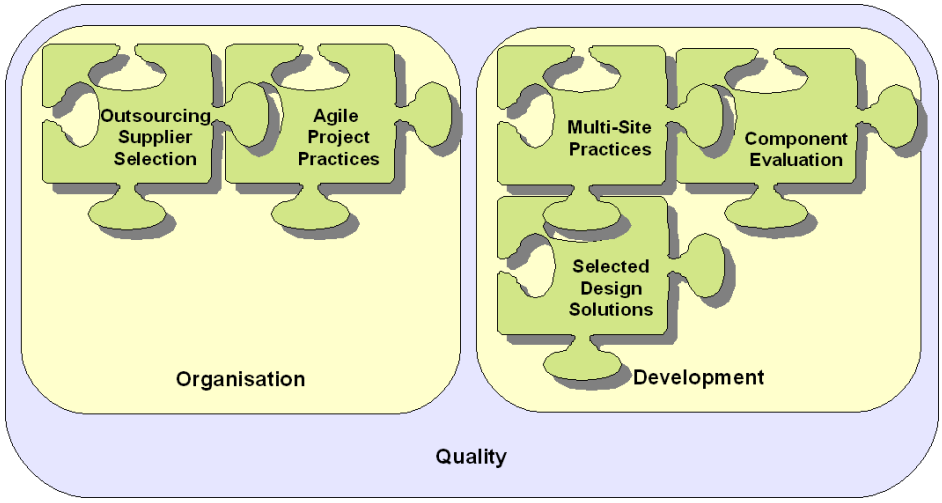



**Figure 1.1** Selected quality improvement areas

All the quality improvement areas are depicted in Figure 1.1, which graphically represents the two area categories in the context of general quality improvement in a software organisation. The selected areas are purposely depicted as pieces of a puzzle, as they were not selected as part of any formal quality improvement programme and they should be regarded as example areas that an organisation can choose to focus on. The areas can be chosen in any order or configuration depending on needs of the organisation that wants to make selective improvements in various areas of their operation.

The five improvement areas can be presented in relation to two dimensions, for example, as depicted in Figure 1.2. The dimensions are created by the Organisation's level of interaction (Intra-organisational, Inter-organisational, and Open to other organisations) and various aspects of Development. Such representation of the improvement areas shows how they relate to the two dimensions and how they may overlap. For instance, area A3 *Multi-Site Practices* spans across Inter- and Intra-organisational levels of organisational interaction as it relates to internal practices usage, but also to cooperation in multiple sites. Additionally, this area spans across Tools, Methods, and Processes in the Development dimension, as the area specifies a specific tool, a process, and a concrete method of process implementation. The boundaries of the areas are very dependent on the context as well as interpretation. Moreover, such a two dimensional representation of quality improvement areas shows clusters of areas that may indicate new areas for improvements in an organisation. Naturally, the view of the improvement areas can be adjusted to particular organisational needs.

Even though the selection of these different quality improvement areas was driven by grass-root level initiatives, these areas can be linked with some mainstream SPI frameworks. For example, the five quality improvement areas can be, at a high level,

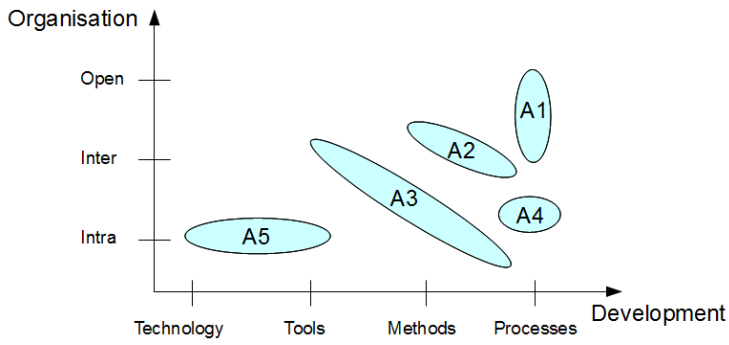



**Figure 1.2** Two dimensions of the improvement areas; *A1: Outsourcing Supplier Selection, A2: Agile Project Practices, A3: Multi-Site Practices, A4: Component Evaluation, and A5: Selected Design Solutions*

linked with some of the 22 process improvement areas defined in CMMI for Development [124, p. 18]. For instance, CMMI for Development specifies *Technical Solution (TS)* [124, p. 456] as an area focusing on designing, developing, and implementing solutions according to requirements. This process improvement area has a similar focus as specified in this thesis' areas in the *Development* category, which groups improvements directly affecting software solution development. Moreover, specific areas of *Selected Design Solutions* and *Component Evaluation* can be seen as parts of the *Technical Solution* process area. This is only an example of possible mapping, however, note that the quality improvement areas discussed in this thesis do not aim at fulfilling any specific quality improvement framework. These small area-specific quality improvements can be regarded as an alternative implementation of Software Process Improvement related to the overall software quality.

## 1.3 Contributions

The main contributions of this dissertation are practical solutions addressing needs of software quality improvements in selected areas of software company operations. The improvements constitute a grass-root level software quality improvement initiative that demonstrates how a small or medium-sized software company may address its software quality concerns. The individual quality improvements that are presented in this thesis have addressed particular problems of software quality, and they gradually improve, directly or indirectly, the quality of the software solution that the software organisation develops. The list of concrete contributions and supporting publications in all five improvement areas are as follows:

- *Organisation - Outsourcing Supplier Selection* focuses on selecting outsourcing partners according to needs of the organisation seeking external resources. In

this area a concrete process for selecting outsourcing, i.e., subcontracting, partners is proposed. It aims at selecting partners who are suitable for work in projects in a software service provider. The process is suitable for and has been evaluated in actual outsourcing partner searches. Furthermore, a general evaluation process with specific evaluation criteria is proposed for cooperation with selected partners. The selection process is based on practical experiences in a case organisation, which have been presented in publication [P4]. Furthermore, the selection and cooperation process was presented in publication [P1], which focuses on overall cooperation with subcontractors.

- *Organisation - Agile Project Practices*, which concentrates on project organisation in specific contexts. In this area first experiences based on 18 project cases are reported [P2]. The experiences compare projects' performance depending on the methodology used. Scrum and non-Scrum projects are compared. Additionally, concrete practices used in Scrum projects are reported. Secondly, based on experiences gained in Scrum projects and the subcontractor evaluation process, a process for cooperation with subcontractors in projects is proposed. The process as well as specific practices are presented in publication [P1].

- *Development - Multi-Site Practices*, which focus on practices that can be used in software development to solve problems encountered in a specific context. As an example, a problem of architectural design decisions assurance in multi-site projects is addressed. A process for architectural decisions assurance is proposed. The process aims at multi-site development projects with outsourcing partners. Furthermore, a tool is proposed for architectural rule assurance in Java applications. The tool has been developed by the author. Both the process and tool were evaluated for their feasibility. The process and the tool are described in publication [P5].

- *Development - Component Evaluation*, which focuses on quality evaluation of third party components that are to be used as an integral part of the built software. The role of such components in the final software solution and the component individual quality attributes affect in different ways the quality of the final software. A framework for quality evaluation of open source components is proposed in publication [P3]. The framework uses a number of selected criteria to evaluate open source components. The framework also views the evaluation in a new way that is similar to construction of software product lines. Finally, results from the framework usage are evaluated.

- *Development - Selected Design Solutions*, where design decisions affecting particular quality attributes are addressed. In this case the dissertation presents empirical findings of design patterns' influence on the performance of distributed systems. The findings are collected for two technology-specific cases presented in publications [P8] and [P6]. Moreover, based on the empirical results, use of an existing tool in a way supporting quality-driven design is proposed. The tool allows a designer to choose a design pattern based on the pattern's influence on particular quality attributes, which is described in publication [P7].

The summary of individual publications' contributions in the different areas of software quality improvements are gathered in Table 1.1. The contribution of each publication in different areas is marked by X.

**Table 1.1** Individual contributions of each publication

| Contribution area / Publication | [P1] | [P2] | [P3] | [P4] | [P5] | [P6] | [P7] | [P8] |
|---|---|---|---|---|---|---|---|---|
| *Organisation* | | | | | | | | |
| Outsourcing | | | | | | | | |
| Supplier Selection | X | | | X | | | | |
| Project Practices | X | X | | | | | | |
| *Development* | | | | | | | | |
| Development Process | | | | | X | | | |
| Component Evaluation | | | X | | | | | |
| Selected Design Solutions | | | | | | X | X | X |

Different research methods for the various contributions, details of which can be found in the publications [P1]- [P8] and in Chapters 2-6, have been used. However, one of the research methods frequently used in this thesis is a *case study*. Runeson and Höst present a case study definition as *'...case study is an empirical method aimed at investigating contemporary phenomena in their context'* [109, p. 134], which is based on definitions presented by Robson [106], Yin [136], and Benbasat *et al.* [17]. Additionally, Runeson and Höst list three reasons that differentiate research methodology in software engineering from other fields. These reasons relate to the focus of the study object that [109, p. 132-133]:

- focuses on organisations developing software,
- is project oriented, and
- investigates advanced engineering work.

This definition reflects the cases presented in this thesis as well as the context of a single company, which is introduced in Section 1.4. Furthermore, the grass-root level quality improvement initiative and the cases discussed in this thesis can be categorised as *action research*. Runeson and Höst refer to differences between case study and action research as: '*More strictly, a case study is purely observational while action research is focused on and involved in the change process.*' [109, p. 134]. Therefore, the presented frameworks or processes that aim at improving specific aspects of organisation functioning and implemented internally in the organisation can be regarded as action research. Additionally, the research findings presented in this thesis mostly aim at improvements or finding out reasons behind specific phenomena. Therefore, the research could be characterised as *Exploratory* and *Improving*, based on Runeson and Höst [109, p. 135], who refer to Robson [106]:

- *'Exploratory-finding out what is happening, seeking new insights and generating ideas and hypotheses for new research'*
- *'Improving-trying to improve a certain aspect of the studied phenomenon'*

In addition to case studies and action research, system development (or constructive research) is used, where a conceptual framework is constructed, system architecture and design are defined, the system is then implemented, and results are analysed [66, p. 635]. Furthermore, the role of a researcher in the grass-root quality improvement initiative can be seen as a stakeholder who actively participates in the improvement process [65, p. 123]. For example, the tool that was constructed for a specific purpose, as presented in Chapter 4, can be seen as usage of system development as a research method. A detailed review of various research methods and approaches based on a number of publications in software engineering has been presented by Glass *et al.* [49].

## 1.4 Context of the Dissertation

This dissertation has been created based on the author's experience in a software house company providing software development services for their customers. The work reflects the author's interests as well as observations made when acting in different roles in the organisation. Those various roles, which included software developer, lead developer, and project manager, allowed to obtain better understanding of the organisation work and possible areas of operation that were good candidates for quality-related improvements. Therefore, the improvement selection was strongly related to the personal roles of the author of this thesis. Furthermore, as the proposed improvements were not a part of any official SPI programme, no other improvement methods have been evaluated for the case organisation. It should be also noted that the recommendations proposed in this dissertation are primarily targeted to medium-sized organisations providing software services. However, other software organisations are expected to find these recommendations useful as well.

The context of this dissertation is software service provider Solita[2], which provides its services to customers in various domains. The domains range from media, through industry, various services, telecommunication, to public institutions. Moreover, from the technology point of view Solita offers expertise in software solutions, business intelligence and data warehousing, and integration solutions. As a software solution provider that works very closely with its customers Solita operates in a specific context that differs from organisations that focus on product development. The main difference is the close cooperation with end customers. Additionally, from organisational and cultural points of view it is worth mentioning that the company is located in two Finnish cities, Tampere and Helsinki. The two offices accommodated in 2009 about 150 software specialists. The company has successfully operatee and grown organically since 1996.

These context details should be important for understanding the background of this dissertation. This dissertation reflects both the personal experiences of the au-

[2] www.solita.fi

thor and the context of a specific company type. Additionally, it is worth remembering that this work has been created using a bottom-up approach by collecting small quality improvements into a bigger quality improvement initative whole at the company level. Therefore, the organisation and content of this work reflect this approach.

## 1.5 Organisation of the Dissertation

This dissertation in the following chapters presents the different quality improvement areas in a software service provider that affect, directly or indirectly, the quality of the final software solution. These areas are presented starting from areas in the *Organisation* category and next areas in the *Development* category are discussed. Therefore, in the category of *Organisation*, an outsourcing supplier selection and evaluation process is presented in Chapter 2. Next, in Chapter 3 selected Agile project organisation practices are discussed and recommended. Then in the category of *Development* selected multi-site development practices are discussed first, in Chapter 4. Next, in Chapter 5 component evaluation from the quality point of view is discussed. Then, specific design solutions are examined in Chapter 6. Each of the chapters presenting the different areas of software quality improvements provides background information on the specific area, the details of contributions, and a summary discussing the given area in the context of overall software quality initiatives at an organisation level. In Chapter 7 related work for each quality improvement area is presented. Finally, in Chapter 8 conclusions of the research are presented.

# CHAPTER 2

# Organisation - Outsourcing Supplier Selection

Outsourcing supplier selection is the first quality improvement area within the *Organisation* category (see Figure 2.1). Organisation-related activities do not directly result in software production, however, as will be presented in the following chapters, they have an important role in the software development processes.

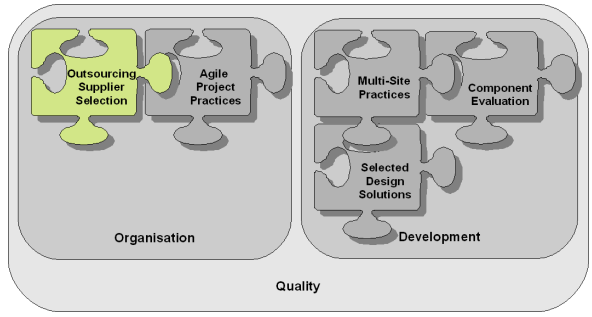



**Figure 2.1** Outsourcing supplier selection

Outsourcing supplier selection is an important activity for software organisations that are planning to cooperate with other software organisations in order to get access to new specialists. There are many reasons why companies use third parties for delegating partly or fully development activities. The main reasons can be lower labour costs and access to skilled specialists [33]. However, a decision of using services of an external party for software development is important from the software quality point of view. The competences as well as process and cultural compatibilities can cause additional challenges that are not present in the case of in-house software development. Therefore, a choice of service suppliers and a way of cooperation with them influence the quality of software produced. This chapter shows, based on an industrial case study, how a software service provider company can select their partners.

In this chapter brief background information of the outsourcing case is presented along with key terms and concepts in Section 2.1. Then Section 2.2 presents how a software organisation can select their outsourcing partners and evaluate their work. Finally, this chapter is summarised with highlights relating to the main thesis questions in Section 2.3.

## 2.1 Outsourcing Considerations - Background

The need for definition and improvements in processes used for outsourcing supplier selection was observed in the case company when it was decided that external developers were needed for software development. Due to a specific set of requirements and at the same time the aim of having a light-weight process, the process was specified in-house. Despite the fact that the selection process was adjusted to the needs of a specific organisation, it still seems generic enough for other software service providers to be able to utilise it in their organisations.

Before going into details of outsourcing partner selection, we need to explain the main terms. The definition of outsourcing is relatively broad and can refer to many activities in an organisation that are passed to another organisation. For example, Pirkko Östring defines outsourcing as follows: *Outsourcing is the strategy of using external companies to provide a service or to manufacture products for your company.* [96, p. 7]. Naturally in the case of a service software provider that seeks specialists who can participate in development of a software solution for its clients, the focus of outsourcing is on service rather than on manufacturing. The practice of using third parties to fulfil one's own contractual obligations is called subcontracting[1]. In later parts of the text terms *subcontractor* or *subcontracting partner* will be used to refer to an other software organisation that provides its specialists for work in projects of the service software provider.

## 2.2 How to Cooperate with Subcontractors?

This section presents a process that shows how a software organisation can select a suitable subcontracting partner and how the organisation can evaluate the cooperation. This process has been used and evaluated at Solita, as was first described in detail in publication [P4]. Later an overview of this process was presented as part of practical recommendations on cooperation with subcontractors in publication [P1].

In order to present how an organisation can cooperate with subcontractors, we first examine the cooperation process in Section 2.2.1. Then we present two phases of the cooperation process, namely, selection in Section 2.2.2 and evaluation in Section 2.2.3. Then we discuss experiences from process usage in Section 2.2.4.

### 2.2.1 Subcontractor Cooperation Process

A software service organisation that closely cooperates with its customers on specific software solutions must carefully choose subcontracting partners. The main reasons for a careful consideration of such partners is the quality of the end solution, which naturally is linked with the technical skills of the developers working on the solution. Additionally, often the partners work directly with end customers of the software service organisation. Therefore the partner's personnel must be familiar with both processes of the software service organisation as well as with the

[1] As per dictionary definition: to subcontract: 'To contract out portions of a larger contracted project.' in `http://en.wiktionary.org/wiki/subcontract`

culture of the organisation. Based on these two requirements it can be seen that a service software organisation may want to create a selection and evaluation process for subcontractors that suits its specific needs.

An overview of the subcontractor cooperation process is presented in Figure 2.2. The process consists of two phases. The first phase is the selection phase, which results in a supplier list. The second phase is the evaluation phase of the work done with subcontracting partners that have been chosen in the selection phase.

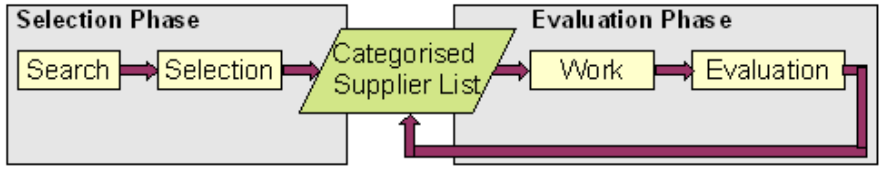



**Figure 2.2** Subcontractor cooperation process overview [P4, p. 226]

The selection phase consists of two main steps: *search* and *selection*. In the search step potential subcontracting partners are searched. Then based on the defined criteria some subcontractors are selected for cooperation. The evaluation phase consists of *work* and *evaluation* steps. The work step represents the actual project done with a subcontracting partner. The evaluation step represents any feedback and lessons learned from the work done. Having gone through the main phases of the process it is now possible to go into details of each phase.

### 2.2.2 Subcontractor Selection

The subcontractor selection process aims at finding a large number of possible candidates and then using various criteria and steps to narrow down the number to a selected group of partners. It should be noted that the process may end without finding any potential cooperation candidate. The process is an additional tool for collecting information based on which decisions are made. However, the criteria used in the process are not strictly defined, so that a small variation from the expected criteria would automatically disqualify the potential candidate for partnership partner. Finally, even the companies that are rejected at the selection phase may be reconsidered if their status changes. For example, if a company grows over time and reaches a size that is appropriate for cooperation, or the competences of a potential partner become in line with the ones that are expected from partners.

The details of the subcontractor selection process are presented in Figure 2.3. The first step is *initial search*. This search is done in order to get as many possible candidates as possible. Various sources are used in the search. Naturally, an Internet search is one source of potential candidates, however, other sources like personal recommendations or organisations helping in contacting businesses can be useful. The *initial search* is typically limited by location, which is decided beforehand, and by technology keywords. The location can be limited to a certain geographical area,

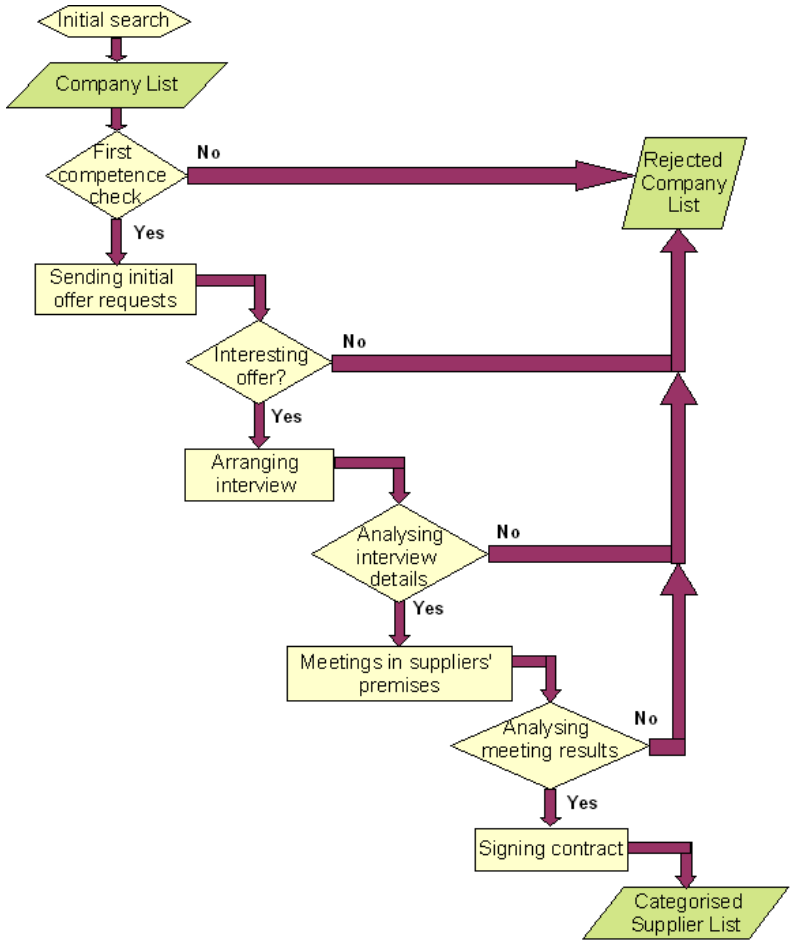



**Figure 2.3** Subcontractor selection phase [P4, p. 227]

e.g., a nearshore location [24], which from the perspective of Finland may mean Central or Eastern Europe. The *initial search* results in a *company list* that is categorised and gradually narrowed down in the following selection steps.

The next selection step is the *first competence check*. This step is designed to filter out companies that do not have even potential qualifications for cooperation. An example of such a situation can be a company that was included in the *company list* based on information from a third-party organisation, but which after a brief check turned out to work in a different IT branch than expected, e.g., the potential partner changed its business from software to hardware, and the recommending organisation was not aware of the change. The *first competence check* is done based on publicly available information, which in most cases is collected from companies' websites. The companies that are rejected are moved from the *company list* to the

*rejected company list*.

After the *first competence check* the potential companies are contacted with initial offer requests in the *sending initial offer requests* step. The request contains questions about interest in providing subcontracting services, general company profile with basic data, and an example pricing model. Based in the provided data the companies that provide *interesting offers* continue to the *arranging interview* step.

The interview is arranged as a teleconference for practical reasons. As a preparation step for the interview the potential subcontracting partner receives a questionnaire that contains multiple questions about their organisation. The questions in the questionnaire are discussed later in this section. The potential partner is asked to send back the answers to the questionnaire before the interview. During the interview the representatives from the software solution provider discuss with representatives from the potential partner organisation. The discussion concentrates on the questions from the questionnaire that may need some clarifications, as well as general discussion about a possible cooperation model.

When the interview stage is concluded, the results of the interviews with potential subcontracting partners are analysed, which is named as the *analysing interview details* stage. It focuses on the information provided in the answers to the questionnaire and in the interview. Based on the results of the analysis, selected potential partners are asked for *meetings in suppliers' premises*. It should be noted that at this stage of the selection process there are only a few candidates for subcontracting partner and it is possible to have face-to-face meetings with all of them. The meeting arranged in the potential supplier's premises aims at discussing details of the cooperation, getting to know each other, as well as see the work environment of the potential partner. Additionally, at this stage interviews with specialists that potentially could work in software service provider projects can be conducted. Finally, after the visits, results are once more analysed (*analysing meeting results*), either the cooperation process proceeds to *signing contract*, or the potential partner is rejected. When a cooperation contract is signed the subcontracting partner is put on the *categorised supplier list*. The list consists of a few partners with whom the cooperation is done and the partner's specialists can be used in projects.

The use of a questionnaire aims at getting understanding on whether a potential subcontracting partner is suitable for cooperation in solution-oriented projects. The details of the selection criteria are presented in publication [P4, p. 228-229] and summarised here. The main areas of interest focus on the size of the potential partner as well as experience in different projects. Naturally, project number, size and roles of the potential partner specialists in the projects are also important, i.e., whether the specialists were technical leaders in their projects. Additionally, the methodology of conducting projects is also asked, as the closer the methodologies and practices of both organisations are the easier it is to cooperate in projects. Finally, the questionnaire contains questions about the customer relationships of the potential subcontracting partner. All this information is collected in addition to typical business and financial information needed in any business cooperation.

When the subcontracting partners are selected normal cooperation is started and projects are realised. The cooperation with subcontracting partners is described in Section 2.2.3. It should be noted, however, that the selection process can be executed

multiple times in one organisation depending on the changes in the needs. For example, if partners with some new technological expertise are needed, a new selection process can be initiated in order to find partners meeting the new requirements.

### 2.2.3 Subcontractor Evaluation

The subcontractor evaluation, which is part of cooperation with subcontracting partners, starts once the selection has been completed and partners to work with in projects have been chosen. Having the partners ready for cooperation they can join a project once there is a new project. The evaluation process is presented in Figure 2.4. The process aims at working efficiently with subcontracting partners and based on gained experience adjust cooperation practices on a way most benefiting the cooperation. It should be noted that changes may be required on both sides, i.e., both on the software service provider and on the subcontracting partner side.

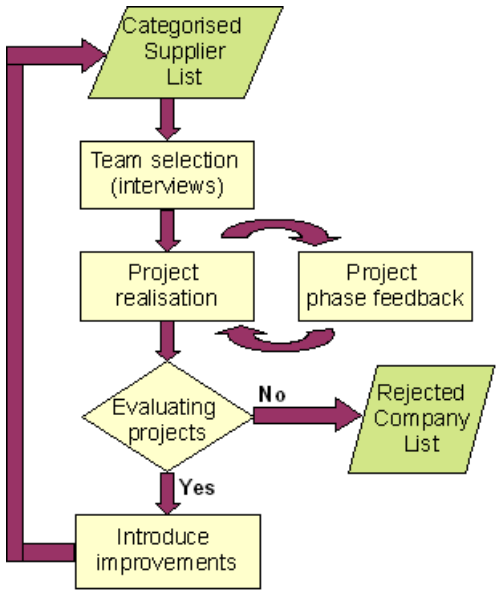



**Figure 2.4** Subcontractor evaluation phase [P4, p. 230]

The first step of evaluation is *team selection*, which is a natural part of any project formation. A new project team can include a few or consist of specialists from a subcontracting partner's organisation. The individuals are chosen based on their suitability for a particular project. Next, *project realisation* starts. The evaluation process does not specify how a project should be organised and according to what methodology. However, also during project realisation *project phase feedback* should be collected. This feedback might cause adjustments to the way the project is conducted. A good process framework for getting frequent feedback during a project is Scrum [111], which in the context of global software development in distributed

teams was studied by Paasivaara *et al.* [98]. Their findings show the positive impact of using Scrum in a distributed team developing a product. Moreover, Paasivaara *et al.* [98] showed how Scrum practices can be used in practice. The details of project organisation and practices including subcontractors' participation from Solita's experiences are presented in Chapter 3 of this thesis.

When a project is finished the results are evaluated (*evaluating projects*). The evaluation is based on a few selected criteria that aim at evaluating the quality of the cooperation and ultimately the resulting software solution quality. For that purpose a few possible criteria and a selected sub-set of them were listed. The evaluation criteria details are presented in publication [P4, p. 231-232] and summarised below. The used criteria were found most feasible and most easy to use. Naturally, typical software metrics, e.g., those presented by Stephen Kan [67] like object-oriented complexity metrics, could be used, too. However, code level metrics used for evaluation of cooperation with subcontracting partners could be too tedious for practical applicability. The metrics that have been selected for evaluation are the following:

- *Profit margin* is used as a metric for a business point of view on the project performance. Even though it is a straightforward metric, it must be applied with care and taking into consideration the context of project realisation. The metric value can be obtained from the company's invoicing system.
- *Communication factor* which is the ratio of time spent on communication to total time spent on project realisation. This metric aims at providing information on possible differences in communication patterns between the projects. The communication in this context should be understood as any formal communication that is recorded in the time tracking system, for example, project meetings, workshops, etc.
- *Percentage of subcontractors* in the project team, which aims at monitoring whether there are any differences between projects organised in different project setups.

Additionally, qualitative criteria of evaluation were used. These consist of project methodology and customer satisfaction. Especially customer satisfaction is essential in the context of a software service provider that works closely with its customers.

The metrics and criteria were discussed in publication [P4] in the context of cooperating with subcontractors. However, the metrics were later used in a broader context of various project organisation models, which was generally discussed in publication [P2] and further discussed in publication [P1]. The discussion on various practices of project organisation and reported results are presented in Chapter 3. A brief summary of selection process usage is presented in Section 2.2.4.

### 2.2.4 Selection Process - Experience Gained

The selection process has been used for selecting subcontracting partners at Solita. The process has been applied at least twice for separate cycles of partner search. The different search cycles differed as for location of the search as well as the technological profile of the potential partners. It should be noted that the search had been limited to nearshore locations, i.e., locations that allow for relatively quick travelling,

a similar time-zone, as well as a relatively similar culture. The main reason of this limitation was the specifics of software service provider work that may require frequent communication with the end customer, for example, a one-day business trip can be requested at short notice.

The results of two selection cycles are summarised in Table 2.1, which shows how the initial number of potential subcontracting partners (Initial Pool) lowered at each selection step. In the first search cycle 26 companies from about 200 were initially selected. From those 26 companies 12 were contacted after filtering out those that had unsuitable profiles. The interviews and the following analysis resulted in visiting five potential cooperation partners. Finally, from that short list a few partners were selected.

**Table 2.1** Subcontracting partner selection cycles

| Search Cycle | Initial Pool | Initial Search | Contacted | Interviewed | Visited |
|---|---|---|---|---|---|
| I | 200 | 26 | 12 | 5 | 5 |
| II | 150 | 77 | 38 | 8 | 6 |

The second selection cycle with location and technical requirements different than those in the first case was conducted. In the second search cycle there were initially 77 organisations out of 150. From the initially selected 38 were contacted and 8 interviewed. Then 6 companies were visited. That search cycle did not result in selecting any subcontracting partners, however, the reasons are not related to the selection process.

The presented results show the each step of the selection process resulted in reducing the number of candidates for subcontracting partner. The selection was based on additional information brought by each of the selection steps. The process is generic and simple and can be used in future partner searches.

The process was found useful and flexible. The differences between the first and second search cycle were only the technology scope and location of the potential partners. Therefore, the process was easy to parametrise. The level of details was found sufficient for making decisions and, at the same time, the selection process was not too tedious to implement. Moreover, the cooperation with the selected partners was successful. It is likely the selection process will be used in the future if a new search cycle is needed.

## 2.3 Summary

In this chapter we have discussed a subcontractor cooperation process that has been tailored to the needs of a software service company seeking subcontracting partners for its projects. The process consists of two phases, namely, selection and evaluation. The selection phase is executed rarely for seeking new subcontracting partners, while the evaluation phase is actively realised for ongoing projects. The evaluation phase aims at improving processes and establishing best practices in the cooperation, as well as evaluating the subcontractor's performance. Metrics and criteria selected

for evaluation purposes were presented. The process was used in the case company for a few years. Based on the results of cooperation with the selected partners it can be said that selection was successful. The cooperation process helped in ensuring the quality of software solutions developed together with subcontracting partners.

The proposed process uses common selection criteria and recommendations for successful outsourcing [33, 102, 104] and adjusts them to the requirements of a software service provider. The presented process is abstract enough to be further adjusted in the context of other organisations, while it is still concrete enough to follow it directly. Therefore, an organisation considering usage of subcontractors is able to build their own process that takes into consideration effects of the subcontracting on the final software solution. For example, it is ensured that the subcontracting partner has appropriate processes, knowledge of technologies needed, and experience in working with end customers. Therefore, the presented contributions show how a software organisation can improve their quality in the area of outsourcing supplier selection by introducing the proposed process. Consequently as the competences of outsourcing partners meet the required level for particular software development, we may assume that also the resulting software solution is improved. This answers the main research question RQ1 within the scope of this area. However, as this link between outsourcing supplier selection and final software solution quality is not directly backed up by a sufficient amount of hard data, the actual impact of improvements on final software solution products cannot be determined.

The quality improvement presented in this particular area of organisation operations additionally demonstrates that an organisation can internally create a feasible process. The presented process was flexibly defined in a response to observed needs without a prior improvement plan determined based on other means, e.g., a formal SPI. The fact that an organisation can itself aim at improving processes in the outsourcing supplier selection area answers research question RQ2 in the scope of this area.

Moreover, the process is also in line with general steps of mainstream acquisition processes. For example, CMMI for Acquisition Primer specifies the Solicitation and Supplier Agreement Development (SSAD) process area [105, p. 26] that has main steps of selection process similar to those described in the selection process. Furthermore, the process area of Supplier Agreement Management (SAM) specified in CMMI for Development [124, p. 439] overlaps with the activities specified in the subcontractor cooperation process. The goal of the SAM process area is '*to manage the acquisition of products from suppliers*' [124, p. 439], where the term *product* refers also to services in this context [124, p. 439]. The SAM area specifies the main elements of the cooperation process, supplier selection and evaluation. Therefore, it is safe to state that by using the proposed subcontractor selection and evaluation process an organisation follows a subset of guidelines specified in mainstream frameworks, which addresses the research question RQ3.

The focus of this chapter was on selection of subcontractors and evaluation of their work. More details of specific project organisation practices and methodologies will be presented in Chapter 3. Chapter 3 will also provide details on practices useful in projects implemented with subcontractors. Therefore, it naturally links to the evaluation process presented in this chapter.

# CHAPTER 3

# Organisation - Agile Project Practices

Agile project practices that belong to the *Organisation* category are the second quality improvement area presented in this thesis (see Figure 3.1). Project practices consist of any practices or methodologies used in projects. The practices used in projects affect the project outcome as they determine how the project is conducted. For example, the scope of project practices includes the way how a project team is assembled, how the team communicates, and what tools and methods are used for progress tracking and knowledge sharing. In this chapter particular focus is put on general agile practices and particularly Scrum methodology [111].

In this chapter practical experiences gained from using Scrum methodology for conducting projects are discussed. Scrum and non-Scrum projects are compared. Additionally, this chapter presents a summary of recommendations and practical tips for conducting projects with subcontractors.

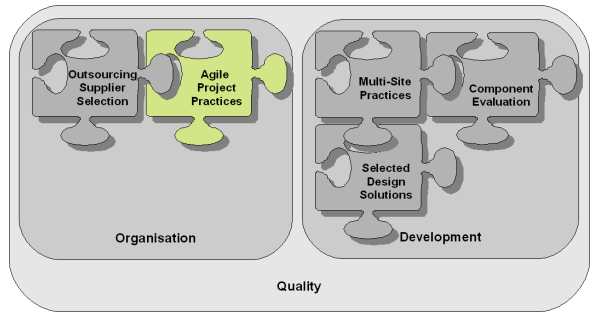



**Figure 3.1** Agile project practices

This area of quality improvement was selected for investigation as the Scrum methodology was introduced to the case company and it was found useful to investigate the impact of the methodology change on projects. Therefore, again the quality improvement area was selected based on emerging needs within an organisation.

In order to provide a good overview of the subject, we first present an overview of selected agile practices in Section 3.1. Then we present contributions that show how a software organisation can use Scrum methodology in Section 3.2. Finally, a summary is presented in Section 3.3, where the contributions are related to the main research questions of this thesis.

## 3.1 Agile Practices Overview

The principles of Agile software development have been defined in Agile Manifesto by Beck *et al.* [16]
*'We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

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

*That is, while there is value in the items on the right, we value the items on the left more.'* The principles of agility in a very condensed form present the values that should drive agile development. However, the manifesto does not specify any concrete methods how it could be implemented. Moreover, under specific circumstances an agile project may need to use some elements of plan-driven practices, as noted by Barry Boehm [18, p. 19]. Therefore, we should briefly discuss one software development methodology that follows the agile principles and was used at Solita. The prominent example is Scrum methodology that has been described by Advanced Development Methods [2] and then by Ken Schwaber [111].

Scrum assumes that certain elements of software development are unpredictable and proposes control mechanisms and risk control remedies to these unpredictable elements [111, p. 10]. Scrum is based on a concept of Sprints that are short development iterations that respond to a changing environment based on empirical feedback [111, p. 10]. The huge difference between Scrum and plan-driven methodologies is that Scrum accepts changes in the project scope until project finalisation. Scrum has been used in Chapter 3 as the main agile methodology. Scrum is only one example of agile development methods. Various agile methods were reviewed by Abrahamsson *et al.*, who also generally characterised agile methods as those that are incremental, cooperative, straightforward, and adaptive [1, p. 17].

Scrum specifies primarily a management method [1, p. 27] but tools and detailed practices, e.g., estimation, are left undefined. The Scrum methodology refers to backlog as a list of items to be developed [111], but the details of a backlog are not defined. Moreover, agile methods may require usage of the same tools that were used in traditional projects, but most likely the tools will be used differently, as Alan Koch noted [74, p. 35]. Therefore, tools and practices used in real-life projects have been investigated, and the results can be found in Section 3.2. The experiences collected from Solita's projects show how Scrum has been used in real commercial projects, as well as what kinds of tools and practices were used to support Scrum projects.

Having gone through agile method principles and a Scrum methodology overview, we can discuss findings on Scrum practices in a specific context of a software service organisation, as well as analyse differences between Scrum and non-Scrum projects.

## 3.2 Why to Use Scrum in Commercial Projects and How to do it with Subcontractors?

In order to answer this question, the contributions have been grouped into two parts. First, the analysis of 18 Scrum and non-Scrum projects conducted in Solita is discussed. The analysis focused on the differences between Scrum and non-Scrum projects in order to find out whether Scrum methodology improves the performance of projects. The project performance was measured based on multiple criteria, which will be described later in Section 3.2.1. However, the core criteria were already defined as subcontractor project evaluation and they were presented in Section 2.2.3 and in publication [P4]. The results of the Scrum and non-Scrum project analysis were presented in publication [P2].

Secondly, a Scrum-based cooperation process with subcontractors is presented. Additionally, practical recommendations for conducting Scrum projects with team members coming from subcontracting partners have been collected. Generally Scrum projects were found to be better performing comparing to non-Scrum projects. Additionally, the Scrum practices encouraged communication within the team as well as with customers. The customer's active participation in the project was also found to have a positive effect on project performance. The recommendations are based on findings from interviews conducted with project managers as preparation for publication [P2]. Furthermore, the findings from publication [P2] are presented as practical recommendations in detail and in a practitioner oriented manner in publication [P1], and summarised in Section 3.2.2. The recommendations are directly usable in a given context.

### 3.2.1 Scrum and Non-Scrum Projects Comparison

The analysis of Scrum and non-Scrum projects was conducted using the Goal Question Metric (GQM) approach [12]. The data for analysis was obtained from interviews with project managers of 18 Scrum and non-Scrum projects, who answered questions corresponding to the metrics defined in GQM and gave free form feedback on the project. Additionally, data, e.g., profitability or project time usage, was obtained from the company's IT systems. The GQM consists of three parts:

- *Goal* that determines the ultimate purpose of the investigation,
- *Question* that asks about a certain aspect of the investigated issue and supports the goal,
- *Metric* that specifies what should be measured in order to answer a specific question.

The goal of this comparison of Scrum and non-Scrum projects was defined according to the GQM template as follows: (Purpose) *Find out* (Issue) *the impact of* (Object) *Agile practices in software projects* (Viewpoint) *from the software service provider company point of view* [P2, p. 224]. In order to reach the goal, five questions [P2, p. 224-225] were defined:

- *Q1: Do the current Agile practices benefit projects?* This question aimed at finding out whether agile practices, and in particular Scrum, were benefiting projects. In order to answer this question, three metrics were used.

  1 Customer satisfaction, which represented the customer point of view on the project performance.

  2 Team performance, which represented the internal point of view of the project manager on team performance.

  3 Profitability, which represented a business point of view of the project performance.

- *Q2: Does the customer's direct involvement in the project benefit project success?* This question aimed at finding out whether different degrees of customer involvement during a project affected in any way the project performance. Three levels of customer involvement were distinguished, from non-involvement, through partial involvement where the customer provided occasional feedback on the project progress, to full involvement that meant active and frequent participation in project meetings and providing feedback. The project performance was measured based on the metrics specified in question *Q1*. All of the three metrics were expressed in a point scale from 1 to 5, where 1 was the poorest result, and 5 was the best result. Both customer satisfaction and team performance were assessed by the project manager of a given project. Therefore the metrics can be influenced by the project manager point of view. The profitability was assessed based on financial data.

- *Q3: Is the communication different in Agile projects?* This question aimed at verifying any potential differences in communication between projects depending on methodology used. Communication was measured using two metrics.

  1 Communication factor, which was specified as the percentage of the time spent by the whole project team on formal communication to all project time. The time-related data was obtained from a time tracking system that had separate tasks for internal or customer meetings, reviews, and similar formal communication-related events; hence it was possible to measure the metric.

  2 Project manager time, which was the percentage of the time spent by the project manager on managerial tasks to all time used by the project.

  Both of the metrics used data obtained from a time tracking system that was used in all projects. Naturally, informal communication was not registered by that tool.

- *Q4: How to adapt Agile practices to suit commercial needs?* This question aimed at finding out what practices are used in commercial projects and what kinds of limitations were observed. The findings were based on interviews with project managers and collected as qualitative data, i.e., comments, observations, and individual recommendations.

- *Q5: What kinds of projects suit Agile practices?* This question aimed at finding out whether there is any particular type of commercial project that best suits agile, or in this case Scrum, projects. The data was obtained from interviews with project managers and was based on analysis of data obtained for other questions.

The most complete data obtained to support all the defined questions and metrics is presented in Table 3.1; subsets of this table were published in publication [P2, p. 225-226] and [P1, p. 19]. The table has more data than what was needed for answering the defined questions. The additional metrics were obtained when the main metrics were collected. This extensive data can be used as supportive and comparative data for the future. The rows of the table are grouped based on the project methodology used. The column abbreviations are as follows:

- *Proj.* contains project ID or a number of projects in a given group,
- *Year* is the year when a project was realised,
- *Size* is the size of a project in man months,
- *Comm. factor* is the communication factor that is defined as the time spent for communication to the total project time,
- *Team size* is the number team members,
- *Sub.* is the number of subcontractors in the project team, and
- *Meth.* is the methodology used in the project. The methodologies were distinguished as *WF* for traditional Waterfall-like projects that were relying on heavier planning ahead, *Iter.* for iterative projects that were not planned heavily in advance but that did not follow Scrum, and *Scrum* for Scrum projects. Other columns are:
- *Cust. satisf.* for customer satisfaction,
- *Profit.* for project profitability,
- *Team perf.* for team performance,
- *Sites* for the number of sites in the project,
- *Cust. invol.* for customer involvement level,
- *Contract model* for the project's contract model that could be fixed, meaning fixed scope and price, or Time and Material (TM), meaning the project is realised until the customer decides that the requirements are fulfilled, and
- *PM time* for Project Manager time that expresses time spent for management to the total project time.

The questions were answered based on the collected data.

**Q1: Do the current Agile practices benefit projects?** The differences between the different project methodologies can be seen in Table 3.1 in columns for customer satisfaction, profitability, and team performance. The results are calculated as average values for each methodology used and as total averages for all project methodologies used. As can be seen the Scrum projects generally had higher scores than projects using other methodologies. However, customer satisfaction in Scrum projects was lower than that in iterative projects. There is no easy explanation for this result. As has already been noted the customer satisfaction was not objectively measured (e.g., by using a survey), and therefore the customer satisfaction data provided by the project managers might be inaccurate. This problem should be eliminated in the future by introducing standard customer satisfaction surveys used to obtain customer feedback directly.

**Q2: Does the customer's direct involvement in the project benefit project success?** The results for various levels of customer involvement in projects are summarised in Table 3.2. The data from Table 3.1 is grouped by customer involvement level and methodology used. Additionally, only relevant metrics are used in Table 3.2 for clarity. The three customer levels full, partial, and none, are marked in the table by *Full*, *Partial*, and *No* respectively. The results for each group represent average values of metrics used.

**Table 3.1** Projects grouped by the methodology used

| Proj. | Year | Size (mm) | Comm. factor | Team size | Sub. | Meth. | Cust. satisf. | Profit. | Team perf. | Sites | Cust. invol. | Contract model | PM time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P01 | 2006 | 40 | 13.00% | 5 | 2 | Iter. | 5 | 3 | 4 | 2 | Partial | Fixed | 2.00% |
| P08 | 2007 | 37 | 27.00% | 7 | 0 | Iter. | 5 | 3 | 4 | 2 | No | Fixed | 13.00% |
| P13 | 2007 | 25 | 37.00% | 4 | 1 | Iter. | 4 | 1 | 2 | 2 | No | Fixed | 16.00% |
| P17 | 2007 | 17 | 14.00% | 3 | 0 | Iter. | 4 | 3 | 4 | 1 | No | TM | 9.00% |
| 4 | | 29.87 | 22.76% | 4.75 | 0.75 | Iter. Result | 4.5 | 2.5 | 3.5 | 1.75 | | | 10.22% |
| P03 | 2006 | 27 | 24.00% | 6 | 0 | Scrum | 4 | 3 | 4 | 2 | Full | TM | 8.00% |
| P04 | 2007 | 56 | 20.00% | 6 | 2 | Scrum | 4 | 3 | 4 | 3 | Full | TM | 9.00% |
| P07 | 2008 | 19 | 30.00% | 11 | 2 | Scrum | 3 | 4 | 4 | 2 | Full | TM | 7.00% |
| P09 | 2008 | 16 | 33.00% | 4 | 2 | Scrum | 1 | 4 | 4 | 2 | No | TM | 18.00% |
| P10 | 2008 | 29 | 19.00% | 11 | 2 | Scrum | 5 | 5 | 5 | 2 | Partial | Fixed | 13.00% |
| P11 | 2008 | 7 | 24.00% | 3 | 1 | Scrum | 4 | 4 | 4 | 2 | Partial | Fixed | 15.00% |
| P12 | 2008 | 10 | 30.00% | 7 | 5 | Scrum | 4 | 5 | 4 | 2 | Full | TM | 14.00% |
| P18 | 2008 | 5 | 45.00% | 3 | 1 | Scrum | 5 | 5 | 4 | 2 | Partial | Fixed | 16.00% |
| 8 | | 20.96 | 28.12% | 6.38 | 1.88 | Scrum Result | 3.75 | 4.13 | 4.13 | 2.13 | | | 12.42% |
| P02 | 2006 | 90 | 24.00% | 12 | 0 | WF | 3 | 1 | 3 | 2 | No | Fixed | 8.00% |
| P06 | 2007 | 73 | 14.00% | 7 | 5 | WF | 2 | 2 | 3 | 2 | Full | Fixed | 11.00% |
| P14 | 2007 | 32 | 39.00% | 3 | 1 | WF | 4 | 3 | 4 | 2 | Partial | TM | 16.00% |
| P05 | 2008 | 13 | 47.00% | 7 | 5 | WF | 3 | 5 | 4 | 2 | Full | TM | 8.00% |
| P15 | 2008 | 2 | 19.00% | 2 | 1 | WF | 5 | 3 | 5 | 2 | No | TM | 7.00% |
| P16 | 2008 | 4 | 16.00% | 4 | 2 | WF | 4 | 3 | 4 | 2 | No | TM | 5.00% |
| 6 | | 35.84 | 26.51% | 5.83 | 2.33 | WF Result | 3.5 | 2.83 | 3.83 | 2 | | | 9.40% |
| 18 | | 27.9 | 26.39% | 5.83 | 1.78 | Grand Total | 3.83 | 3.33 | 3.89 | 2 | | | 10.92% |

As can be seen from Table 3.2 projects that did not have any customer involvement during the projects scored the lowest scores. Projects with partial customer involvement received the highest scores. Moreover, there was only one Scrum project that did not have any customer involvement, which also had the worst customer satisfaction level recorded. Therefore, based on the results and differences between projects with and without customer involvement, it can be said that customer feedback improves project performance. However, it is difficult to definitely state to what extent customers should be involved in projects. That may need case by case considerations.

**Table 3.2** Customer involvement in projects [P2, p. 225]

| Number of projects | Methodology | Customer satisfaction | Profitability | Team performance |
|---|---|---|---|---|
| 4 | Iterative | 4.5 | 2.5 | 3.5 |
| 8 | Scrum | 3.75 | 4.13 | 4.13 |
| 6 | WF | 3.5 | 2.83 | 3.83 |
| 18 | Grand Total | 3.83 | 3.33 | 3.89 |

**Q3: Is the communication different in Agile projects?** The communication differences in projects depending on the methodology used can be seen directly in Table 3.1. The communication factor is the highest for Scrum projects, lower for traditional projects, and the lowest in iterative projects. The highest communication in Scrum projects was expected, as Scrum has a few formal elements promoting communication, i.e., daily meetings, planning and demonstration meetings. It cannot be stated certainly why traditional projects had a relatively high score. However, one explanation could be that in traditional projects the amount of formal documentation and planning generates enough communication recorded in the time tracking system to receive a higher value than iterative projects, which do not produce so much documentation.

The project management time was highest for Scrum, lower for iterative projects, and lowest for traditional projects. It should be noted that in the investigated projects the project manager typically also played role of Scrum master and therefore participated in all Scrum meetings. That is one possible reason why Scrum projects received the highest project management time score.

Generally, the obtained results were not significantly different between each methodology used. The differences were within a few percentage points. Therefore it can be stated that Scrum projects tend to spend more time on communication-related tasks. However, the difference comparing to other project methodology types is not significantly higher. Moreover, taking general Scrum project performance into consideration, it can be said that the additional time spent on communication brings visible positive results to projects.

**Q4: How to adapt Agile practices to suit commercial needs?** This question was answered based on comments received from various project managers, who were interviewed on practices used in their projects, as well as general project practices used in Solita. The listed practices include the practices that were used most commonly and that were regarded as most beneficial for projects. A short list of these practices was first published in [P2, p. 226], and later these practices were discussed in detail in the context of using subcontractors in [P1]. In this section the main practices are discussed and their usage together with selected practical recommendations are further discussed in Section 3.2.2.

The practices and tools that have been commonly used in Solita's projects are:

- *Scrum meetings* - short daily meetings where all team members tell the statues of their work. Additionally, design clarifications were made at these meetings. Any discussions that required much time and were out of the scope of a short daily meeting resulted in a separate discussion or workshop.

- *Product Backlog* - product and sprint backlogs were used to keep track of main features planned, ongoing, and done. It was used at Sprint planning. The feature sizes were expressed differently in different projects ranging from hours to abstract units, e.g., story points or Ideal Engineering Hours.

- *Sprint length* - the Sprint lengths ranged typically from two to four weeks. In some projects this was derived from the experience that six-weeks sprints are too long, while two weeks for smaller teams and three weeks for bigger teams are best suited.

- *Planning sessions* - sprint planning was typically conducted by the whole team and in some cases with the customer's active involvement. When the customer was not involved, the customer point of view was represented by somebody else, usually the project manager.

- *Sprint demonstration and retrospective* - were often combined in one meeting with different parts. The customer was not always involved in the demonstration, but it was noted that such participation was very useful and could have helped in certain cases to avoid mistakes. Also teams were better motivated to keep up high quality when the customer participated in demonstrations.

- *Customer involvement* - in general frequent customer feedback seemed to be a desired element of a project, which however could not be realised in all cases. No criticism was recorded on customers being actively involved in projects, while there were recorded comments on too little involvement of customers.

- *Continuous integration* - was used to perform software builds and provide feedback to the whole team on the status of the software.

- *Automated tests* - test levels were different depending on the project. However, at least at the unit test level automated tests were dominant. Moreover, in some cases other test types were executed for each build.

- *Collaboration tools* - were used to share knowledge and foster communication in the project teams. Various collaboration tools were used, including *Instant Messaging* used for everyday communication, *Issue tracking system* used for progress tracking, *Wiki* for light-weight documentation creation as well as any project instructions.

- *SVN tagging/branching* - version control systems were used to keep track of changes in the software produced. Also tagging and branching were used to separate code at different stages of development (released versions and current development).

- *Documentation* - was present in all projects, but the extent of the documentation differed significantly from project to project. Typically, general system-level architecture was created at the beginning of a project. The documentation evolved during the project and more details were added as the implementation progressed. In some cases simple wiki pages were found sufficient, but in other cases traditional documentation was produced as well. The level of documentation in a project often depended on the customer requirements to have or not to have formal documentation.

It is important to note that even though the main elements of Scrum (e.g., scrum meetings, backlogs, self organising teams, shared code ownership, continuous integration, short development intervals) are used at Solita, one could argue that this is not real Scrum, as some constraints are imposed by the commercial context, e.g., a fixed scope of a project. Therefore, Scrum at Solita could be understood as Scrum-like practices, where the main methodology elements are used but not always all of them. In any case, the provided list of practices shows which elements of Agile practices, and generally good practices, were considered as important and useful.

**Q5: What kinds of projects suit Agile practices?** This question aimed at finding out whether Agile practices should be used with any particular types of projects. The question was answered based on comments from interviewed project managers as well as results gathered in Table 3.1. Most of the projects that were analysed in Table 3.1 were software implementation projects. However, there were also data-warehousing projects that may require much more scoping. As can be seen in Table 3.1 two contract types were used, i.e., fixed scope and time and material (TM). The TM projects were more common in Scrum projects, while fixed scope contract models were more common in traditional projects. Even though TM contract models are very suitable for Scrum projects as they encourage the customer to actively participate in projects, the Scrum projects that used a fixed scope were successful as well.

Therefore, based on the collected data and comments from project managers, a recommended project should preferably be a development project with a time and material contract model. However, there are no practical limitations on project scope or contract model. As was presented also fixed scope projects can successfully use Scrum.

**Other results.** In addition to the answers to questions Q1-Q5 it is possible to briefly discuss other findings based on data from Table 3.1. By analysing the changes over time it can be seen that the number of Scrum projects increases with time from one project in 2006 and 2007, and 6 in 2008. On the other hand, iterative projects were on a decrease and none were realised in 2008. Interestingly, traditional projects (WF) are at a relatively low level through the years, but they are on a slight increase between 2006 and 2008. One explanation could be that the projects' nature required detailed planning ahead and Scrum was not considered, or Scrum was not considered in the first place.

Furthermore, based on the data in Table 3.1 it can be seen that most of the projects that were investigated had an average size of 27 man months. The project teams on average consisted of five team members, from which about two were subcontractors. Finally, most of the projects were realised at two sites, with or without subcontractors. The multi-site projects without subcontractors were realised in two company locations in Finland.

Having answers to all the defined questions it was possible to conclude that Scrum projects showed a positive influence of the methodology used on project performance. Additionally, certain elements of Scrum, particularly communication and customer involvement, positively impact projects as well. Generally Scrum seems suitable for any project type. However, implementation projects and in particular projects with a time and material contract type were especially suitable for Scrum.

### 3.2.2 Recommendations on Subcontractors in Scrum Projects

The experiences from using Scrum in projects where subcontractors were present led to collecting a few practical recommendations on how to organise projects with subcontractors. The recommendations include practices as well as tool and were presented in publication [P1]. In this section an overview of those practices and selected practical recommendations are presented. The practices are discussed around three project phases. The phases are:

1 *Preparation*, where the project is organised including team members selection,

2 *Development*, where the software solution is implemented, and

3 *Release*, where the developed solution is released to the customer.

The *preparation* phase is an organisational phase that focuses on setting up the project. The project scope is usually known, at least at some level, based on customer requirements and project order. Based on the more or less detailed requirements also the estimated project size is known. Based on the information it is possible to decide the needed team size and competences of subcontractors. The subcontractor list including competences is available as a result of the selection process presented in Chapter 2. When assembling a team that is going to be distributed across sites it is good to balance the sites so that there is no site with only one developer. Unbalanced sites may result in isolation and deterioration in communication between the sites.

When a team is assembled all team members should receive access to all tools and resources needed for the project regardless of their location or organisation. The

team's first activity is an initial project meeting (i.e., kick-off meeting) where the project scope, objectives, and procedures are explained. If the team members are not experienced in Scrum methodology, the methodology is explained as well. After the preparation phase is completed the development starts.

*Development* follows the principles and activities of Scrum methodology. An overview of Scrum activities within a Sprint is depicted in Figure 3.2. Each Sprint starts from detailed planning of selected features for the given Sprint. The features are chosen by Product Owner, who in many cases is the same person as the project leader. However, also the customer decides on the Sprint scope if the customer is actively involved in development. Based on the planning Product and Sprint backlogs are updated. Sprint tasks are placed in a task tracking system. Development progress is followed up at daily Scrum meetings where ongoing work, planned work and possible impediments are reported. The Scrum meetings are led by a Scrum Master who facilitates the meetings as well as removed impediments.

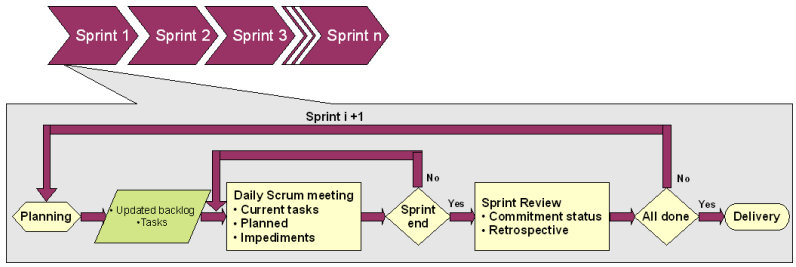



**Figure 3.2** Scrum activities overview [P1, p. 8]

A Sprint ends with a Sprint Review where the work done is presented to the customer, in an ideal situation, or at least to the team. The customer's, or the team's, feedback decides whether corrections are implemented. Moreover, at the end of a Sprint a retrospective on the Sprint is done with the whole team. This is a chance for the team to discuss any positives or negatives of the Sprint and agree on improvement actions. This is additional feedback in addition to the feedback provided by the customer regarding the implemented functionality. If all functionality is implemented in a Sprint, the development ends. If some functionality is left, a new Sprint starts with a planning session.

The tools and practices listed in question Q4 on page 32 are used during the development phase. Particularly the Collaboration tools are used on a daily basis. Good communication is essential for a project's success, and is of even greater importance when the development team is distributed. A practical tip regarding communication practices was given in publication [P1, p. 9]: *'A development team can use a common chat where all the team members can easily exchange options, notify about changes, ask questions, and see availability of other team members. A chat can be used more willingly by subcontractors from other locations than phone, as it is less disruptive and more cost-effective. Team chat rooms have proved to be quite useful in real life projects.'*

Other tips provided focus on the particular context of working with team members from another organisation. The nature of these tips can be summarised as strong emphasis on equal and easy access to all tools, which in many cases means web-based applications, and secondly promotion of good communication in the team regardless of the team members' location. Hence the tip about a common chat that aims at replacing informal communication that happens naturally in collocated teams.

*Release* is the last phase of a project. This phase starts when the development is completed. Release focus is on handing over the software solution to the customer. Final tests are possibly executed in this phase unless all tests were automated and executed during the development. Finally, the customer may need support in deploying the solution. In any case the release delivers the whole software package, documentation, and test reports to the customer. When all these activities are performed and no faults are found in testing, the project activities end. The very last activity, before the project ends and the team is disassembled, is a summary meeting where the project achievements and lessons learned are discussed.

## 3.3 Summary

In this chapter we have presented results of analysis of 18 industrial projects and compared Scrum and non-Scrum projects. The results showed that generally Scrum projects had higher profitability and team performance, which is the main reason why Scrum should be used in similar contexts of commercial projects. Additionally, customer involvement helped projects in achieving their goals. However, the extent of customer involvement should be decided case by case. It should also be noted that the presented data came only from one organisation and that the data was collected from a limited number of projects, i.e., 18 cases. This collection did not include equal representation of different project types, namely, six traditional project types, 8 Scrum projects, and four iterative projects. Therefore, the data amount does not seem sufficient or representative enough to statistically determine the extent of individual project methodology on project performance. Still, this data is useful as informative background and encouragement for other organisations to collect additional data for comparison.

Furthermore, a list of practices and tools that were found useful and beneficial for the projects was collected. This list was further described in the context of working with subcontractors. Generally work with subcontractors does not differ significantly from working with team members coming from the same organisation. However, first of all the subcontractors were selected, as presented in Section 2.2.2, in a way that ensured that they were familiar with technologies and a similar work environment. Secondly, the tools and practices selected focused on equal access to the work environment for all team members, good communication, and frequent feedback channels used to receive feedback from the customer and the team. All these results and practical pieces of advice show how a software organisation can work in projects with their subcontractors using Scrum methodology. The possible improvements in project performance can be also indirectly linked to improvements

in the final software solution, which addresses the research question RQ1 in respect of this improvement area. Again in the case of improvements in the *Organisation* category, they support software development, but do not directly affect the software. However, based on criteria like customer satisfaction or profitability, it is possible to assume that customers were more satisfied with the software they received, which is an outcome of a software project. Furthermore, the profitability level can also be linked with software quality through possible rework done under warranty.

The software quality improvement is achieved in this area of project practices by using development methodology, i.e., Scrum, that improves project performance. The comparison of results in Scrum and non-Scrum projects shows a positive impact of the methodology used in the project. Additionally, needs were identified for better measurement of customer satisfaction as the current, subjective method, may not produce convincing enough results. Furthermore, concrete recommendations on ways of working with subcontractors in Scrum teams were provided. The recommendations were based on practical experiences. Therefore, software organisations working in a similar context can improve their quality by introducing the recommended practices in their projects, which addresses research question RQ2.

The presented practices and ways of cooperating in projects with subcontractors best fit the process area Integrated Project Management (IPM) +IPPD defined in CMMI for Development [124, p. 145]. The purpose of IPM has been defined as '*to establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process that is tailored from the organizations set of standard processes*' [124, p. 145]. The usage of Scrum methodology that encourages involvement of multiple project stakeholders can be seen as following the purpose of the IPM. Naturally, the usage of Scrum alone does not fulfill all requirements defined in the IPM. However, based on the IPM process area purpose being in line with the Scrum usage purposes, and literature reports on CMMI and agile method mappings mentioned, the project practices presented in this chapter do not seem to prevent possible adoption of a mainstream SPI framework. The ability of a software company to choose Scrum as its project methodology and still being able to aim at fulfilling requirements of a mainstream quality improvement framework addresses the research question RQ3. The choice of agile project practices does not mean that the software organisation has to abandon a possibility of introducing a fully-fledged SPI framework in the future, which leaves flexibility in choices for the organisation. Naturally, the presented project practices do not fulfill fully CMMI for Development requirements. However, that was not the aim of these project practices.

# CHAPTER 4

# Development - Multi-Site Practices

Multi-site development practices is the third area of quality improvements, and the first one in the *Development* category that is presented in this thesis (see Figure 4.1). The term *practices* in this chapter refers to any practices or processes that are used for software development. The practice presented in this chapter particularly focuses on multi-site development. Multi-site development is an example of a specific context for development. Depending on a particular context, the development practices may need adjustments to make them suitable for the context. Therefore, in this chapter a specific practice for multi-site development is presented.

In this chapter we start by very briefly discussing the background of the multi-site development case in Section 4.1. Then a process for ensuring architectural decisions and an example tool supporting the architecture validation are presented in Section 4.2. Finally, the contributions and their relation to the main thesis questions are summarised in Section 4.3.

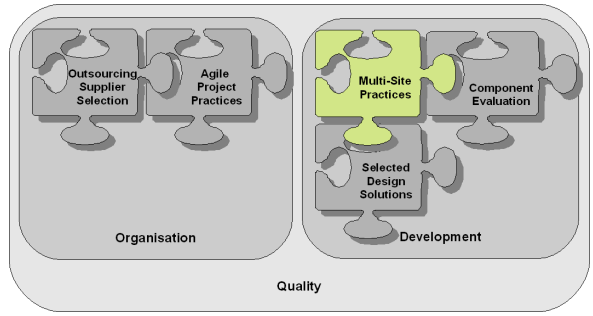



**Figure 4.1** Multi-Site practices

## 4.1 Need for Multi-Site Development

The need for additional improvements in development processes was observed when the development in the case company became distributed. The distribution was between the locations within the organisation, as well as locations of subcontracting partners. Therefore, the improvements proposed in this chapter are a response to changes in the environment in which the organisation operates. However, distributed development is a common practice and the proposed improvements can be applicable to any organisation working in a similar environment.

Multi-site development is a variant of software development that is relatively

common in today's development. The goal of the development regardless of the physical arrangements is the same as in single-site development - simply create software according to the requirements. However, the fact that the development is carried out in multiple locations may impact the everyday communication and there can be a need for practices that are specific to the environment. Therefore, in this chapter we focus on a particular problem of ensuring architectural conventions in a multi-site development environment.

## 4.2 How to Ensure Architectural Conventions in Multi-Site Development?

In order to understand a solution to the problem we need to first discuss a process that is proposed (Section 4.2.1) for ensuring architectural conventions in a multi-site environment. Next, we propose a tool that was developed for the process in Section 4.2.2. Then the tool and process are evaluated in Section 4.2.3. The contributions to this area of software quality improvements have been presented in publication [P5]. There are two main contributions. First, the contribution is a process for architectural decisions assurance in a multi-site project setting. Secondly, a tool for this architectural design decisions validation has been developed and evaluated.

### 4.2.1 Architecture Assurance Process

The context for the contributions is a project with multiple locations, at least two sites. Additionally, as it often happens in current development projects, one, or some, sites represent different organisations than the one that designs the software [132]. For simplicity of the project setup it can be assumed that there are two separate sites in two different organisations. One organisation is an organisation designing the software, but it is the other organisation that implements it. In such a context it is not easy to make sure that original design decisions are well communicated to the implementers and that the architectural rules are followed during the development. Even in locally developed systems original architectural decisions can be altered with time, which for example can be addressed by Architecture Constraint Language (ACL) as presented by Tibermacine *et al.* [125]. Furthermore, the two organisations may have different development processes and tools. Therefore, in publication [P5] we presented a process that assists architecture rules validation in a multi-site environment.

Architectural rules can represent a specific architectural style, for example, a layered architecture [22, p. 31]. In such architecture the concept of layers can be disturbed if certain layers refer directly to others bypassing intermediate layers. In order to improve communication between the sites involved in software development and at the same time not require major changes in the individual development practices at the sites, a solution based on profiles is proposed. Please note that here profiles do not refer to UML profiles [36] that are an extension mechanism for a specific technology, domain, or methodology. An overview of the proposed process is depicted in Figure 4.2.

The process starts when architects or any other competent representatives from the two sites meet in order to agree on the main architecture. The two different sites can be referred to as *creation team* (site A) and *validation team* (site B), as depicted in Figure 4.2. The result of the meeting should be a high-level architectural decomposition. That decomposition can be a simple list of main components and it can be referred to as *abstract architectural profiles*. The *abstract architectural profiles* are the base for creation of other profiles used in the development process. There are two more profile types used. One type is *creational profiles* that contain rules on which the software is created, and *validational profiles* that are used to validate the actual system against the architectural rules.

The *creational profiles* are dependent on the tools and methods used by the *creation team*, which gives them the freedom of choice. For instance, *creational profiles* can be expressed as a UML diagram, e.g., class diagram or sequence, and provide enough details to implement the system. However, regardless of the creation specifics, there are a few assumptions made on the creation that need to be fulfilled for the process to work. The first assumption is that creation uses the system decomposition defined in *abstract architectural profiles*. Secondly, the creation can provide details of fine-grained decomposition, namely component-package mappings. Finally, the assumption is that the creation results in code that represents the system designed.

The *validational profiles* are derived from the *abstract architectural profiles* and they define rules of component interactions as specified in the high-level architecture. The *validational profiles* can focus only on the component interactions that are most crucial from the architectural point of view or define all possible component interactions. For example, the architecture can mandate that certain components may not access other components directly, e.g., to keep a separation of concerns of components and the predefined architecture of a system. In any case, the creation team has the freedom of implementation within the limits of the agreed architecture. The *validational profiles* are used for validation of architectural rules against the actual implementation. As *validational profiles* define only components, the validation team needs to know the mappings between components, and packages belonging to that components. As was specified for *creational profiles* the fact of using packages for organising the system is one of the requirements imposed on the creation. Once the

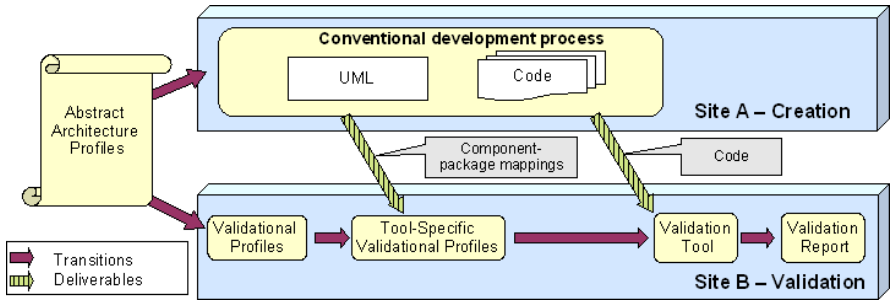



**Figure 4.2** Architecture rules assurance process overview [P5, p. 340]

components are mapped to package sets, they can be used to perform validation of architectural rules against the system (i.e., code) using a validation tool. The result of the validation provides a validation report that summarises the architectural rule fulfillment, as presented in Figure 4.2.

Overall the process requires a certain level of formal communication between the two teams. The creation and validation teams have to communicate at the beginning of the development process in order to define the high-level architecture as *abstract architectural profiles*. Then the creation team has to provide the package names that they use for specific components as component-package mappings that are used in *validational profiles*. The component-package mapping is likely to be incrementally updated while the system implementation progresses. Finally, both of the teams should have access to and visibility of the validation report. The validation report is another channel of communication between the teams, and if there are any misunderstandings the teams can agree and either clarify the implementation that does not follow the defined architectural rules or refine the architecture. The report should be created along with the build process whenever the system code-base is updated.

Practically, the mappings between the components and packages can be stored in an XML file that also contains validational rules. The file can be shared between the sites. As for the validation tool, it needs to be able to use the *validational profiles* and system code as inputs and produce a validation report. An example of such a tool is presented in Section 4.2.2. However, from the general process point of view any tool can be used that fulfils the requirements. The *validational profiles* can be stored from the beginning in a format supported directly by the tool, or in a generic format that is converted into a tool-specific format.

### 4.2.2 Architecture Rule Analyser Tool

A validation tool that can be directly used in the process of architectural rules validation is the Architecture Rule Analyser (ARA) [1]. ARA was designed and implemented in Java by the author of this thesis. ARA is a tool developed for analysing architecture rules in Java. ARA allows for defining custom dependency rules between Java packages and validating those rules based on binary code of the developed software.

ARA is meant as a tool supporting development by validating specific architecture rules during the software development. ARA can provide feedback on rules fulfilment as a part of the Continuous Integration process, which was presented for example by Martin Fowler [45]. An overview of the ARA-based validation process is depicted in Figure 4.3. ARA operates on Java packages, which group Java application classes in logical groups. The packages, or their structures, can comprise components of the software system. The components can have specific relations between each other that are dependent on the chosen architecture style. The style can be, for example, a layered architecture [22, p. 31]. In that architecture style components from one layer may or may not refer to components from other layers, for example, components from the view layer can refer to the subsequent service layer, but they should not directly refer to the persistence layer. ARA is able to detect such rule violation based on the defined rules.

[1] `http://open.solita.fi/projects/ara/`

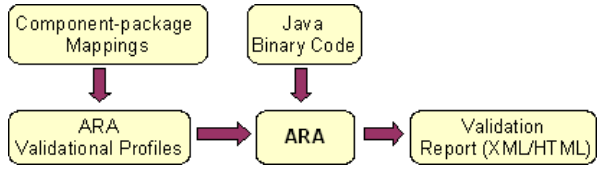



**Figure 4.3** ARA rule validation [P5, p. 341]

As presented in Figure 4.3, ARA uses component mappings, which define packages comprising specific components. Then the components are used in *validational profiles* for defining rules governing component interactions. The *validational profiles* are expresses as a simple XML file, which is human readable but also directly used by ARA. Those rules are then used by ARA to validate Java Binary Code. ARA first goes through all classes of the system and collects package names that are used in each class within a Java package. The information about used packages is collected using Apache's Byte Code Engineering Library [2]. Then the package usage collected from binary code is compared with package definitions defined in *validation profiles*, which contain component-package mappings. The validation results are provided as a summary of valid or broken rules.

ARA supports two types of rules:

- *cannot use* rule that specifies which components cannot refer to which ones, and
- *must use* rule that specifies which component must refer to other components.

The *cannot use* can be used, for example, in the case of layered architecture where components in one layer can or cannot refer to components in other layer. On the other hand, the *must use* rule can represent a requirement of references between components. For example, some components can be required to use logging service. The rules can be freely defined to all or some components, as long as the components can be defined as groups of Java packages. Therefore, the rules are applicable only at the package level. Additionally, the references between packages are recognised as any usage of classes or interfaces from another package. ARA does not analyse references between particular classes or methods.

When ARA is used the architecture rules are defined in an XML file. The file has two main sections, one with component mappings to Java packages, and the second one with actual rules. An example ARA rule file is presented in Listing 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<validationConstraints>
    <!-- Definition of a mapping between
    an abstract subsystem and one real subsystem. -->
    <componentMappings id="analyser">
        <name>Analyser</name>
```

[2] http://jakarta.apache.org/bcel

```
        <component>fi.solita.open.ara.analyser</component>
    </componentMappings>
    <componentMappings id="converter">
        <name>Converter</name>
        <component>fi.solita.open.ara.converter</component>
    </componentMappings>
    <componentMappings id="data">
        <name>Data</name>
        <component>fi.solita.open.ara.data</component>
    </componentMappings>
    <componentMappings id="tools">
        <name>Tools</name>
        <component>fi.solita.open.ara.tools</component>
    </componentMappings>
    <componentMappings id="xml">
        <name>XML</name>
        <component>fi.solita.open.ara.xml</component>
    </componentMappings>

    <!-- Rule definitions  -->
    <rule type="cannotUse">
        <source>converter</source>
        <target>analyser</target>
        <target>tools</target>
    </rule>

    <rule type="mustUse" strength="3">
        <source>analyser</source>
        <target>converter</target>
        <target>data</target>
    </rule>
</validationConstraints>
```

**Listing 4.1** Example ARA validation rules

In this ARA rule example file components and their mappings to real packages are defined. For example, the first component *analyser* consists of one Java package *fi.solita.open.ara.analyser*. It should be noted that ARA component definitions support wild-card character in package name, which allows for flexible system evolvement without a need for mappings changes providing that the naming conventions are followed. Based on the component definitions validation rules are created. In the example ARA validation rules listing there are only two rules, one *cannot use* and one *must use* rule. The *cannot use* rule specifies that the *converter* component cannot refer to *analyser* or *tools* components. If any classes from packages constituting the *converter* component use (e.g., method calls or package imports) any classes from packages constituting the *analyser* or *tools* components, the rule is broken. The validation results are presented in a report, which is in an XML format and can therefore

be easily converted, for example, to HTML. An example ARA report in an HTML format is presented in Figure 4.4. Violated rules are indicated in red (dark gray) colour.

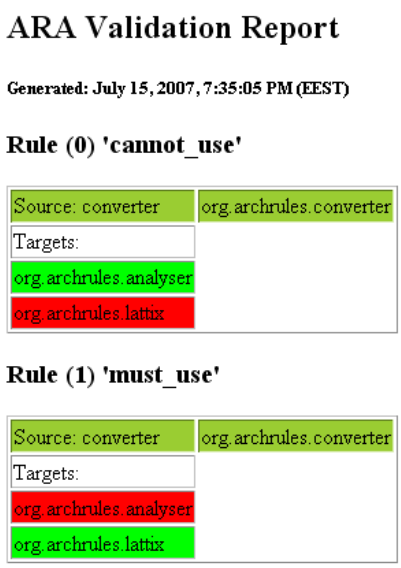



**Figure 4.4** ARA example report

ARA can be easily integrated with build systems that use ANT [3] or Maven [4]. Once the architecture rules are defined, a system can be modified depending on the system evolution. The XML format of the ARA rule file is human readable and it can be edited with a simple text editor.

### 4.2.3 Evaluation

ARA has been used in an example industrial case, as it is described in publication [P5]. First, ARA was used in an illustrative example to validate architecture in a demo application developed at Solita by all new personnel. The application was a simple system using design principles used in the company. The validation was performed in a few implementations of the same application. In addition to the illustrative example, ARA was used to validate architecture in a commercial application developed in a multi-site project. Due to time constraints ARA was applied to existing system versions, which correspond to real-life system development. The results showed that in the initial version of the validated system that consisted of 521 classes in 56 packages the validation took only a few seconds and detected 21

[3] http://ant.apache.org/
[4] http://maven.apache.org/

*cannot use* rule violations and 55 *must use* rule violations. The next major version of the system consisted of 611 classes in 60 packages and 25 *cannot use* and 55 *must use* rules were violated. The number of violations in that case was not as important as the fact that the tool could be feasibly used. If ARA had been applied in a project with ongoing development, the violated rules naturally would have sparked a discussion about the implementation. However, in this case the rules have been defined and applied after the project developed the major version and therefore there was no place for discussion on possible faulty rule definition or implementation.

ARA has been developed in Java and it is freely available as an open source project. ARA provides less functionality than commercial Lattix, which is a fully-fledged architecture analysis tool. On the other hand, the open source Macker tool provided similar capabilities as ARA, but it does not support *must use* rules. Despite its shortcomings in usability and report visualisation, ARA already in its current form is ready for integration with a build system. ARA demonstrates how a tool can support the validation process of a design decision in a software system under development in a multi-site environment. Naturally, ARA can be used as well in single-site development projects even though it was developed with a multi-site environment in mind.

## 4.3 Summary

In this chapter we have presented a process for architectural rules assurance in a multi-site project setting. Moreover, a tool, ARA, was presented that has been developed and implemented in order to fulfill requirements of architectural rule validation. The presented process, on one hand, ensures that high-level architectural decisions are fulfilled during the development. On the other hand, the process allows for relative independence as for the way the creation of the software is done, especially when the creation is done in another organisation. This process provides enough flexibility, yet it ensures the architecture design throughout the development process. Furthermore, the process promotes communication by requiring the different sites to communicate details of the implementation, as well as implementation results as binary code, while sharing the result of the validation. Finally, the presented validation tool has been applied successfully to commercial software.

It has been presented how by using this process and tools proposed a software organisation can improve the quality of developed software by ensuring that architectural rules are followed. The presented development practice is focused on a specific context of usage in multi-sided projects. However, other practices can be developed for other specific organisation contexts. As has been demonstrated, even when different organisations are involved, the process and tools can be set in a way that does not require many changes in organisations' internal processes, at least in specific contexts. Therefore, the contributions of this chapter address the main research question RQ1 by showing how quality improvements can be done in the area of specific practices in development. In this case the improvement directly affects the software produced, as architectural conventions are part of the software produced. As in other cases a limitation of this particular improvement is that it has not been

used in multiple cases, which prevents from generalising conclusions on how much ensurance of architectural conventions improves the quality of the final software solution.

Additionally, this process and tools demonstrated how a software organisation can develop internal quality improvement processes based on their needs and experiences, without explicitly following any SPI frameworks. The specific process and tool for ensuring architectural rules have been developed as a response to needs observed in the multi-site development. Therefore, also the second research question RQ2 is addresses by the presented contributions in this improvement area.

Even if a software organisation does not explicitly follow any SPI, the in-house improvements may be in line a with mainstream SPI. In this case of a specific development practice, the proposed improvements can be seen as part of the Verification (VER) process area in CMMI for Development [124, p. 496]. The purpose of the VER process area is '*to ensure that selected work products meet their specified requirements*' [124, p. 496]. Moreover, VER is specified as an incremental process occurring throughout the development process [124, p. 496]. In that context the architecture rule assurance is a process verifying basic architectural requirements defined for a software solution. Therefore the proposed process of architecture rules verification can be seen as a small part of the larger CMMI's Validation improvement area, which partially answers the research question RQ3.

# CHAPTER 5

# Development - Component Evaluation

Component evaluation is the fourth area of quality improvement presented in this thesis (as can be seen in Figure 5.1). A software system can be implemented as a completely new solution, but in many cases certain components of a larger system can be reused. Ready-made components can come from various sources, for example, in-house components, Commercial Off-the-shelf Software (COTS) components, or Open Source Software (OSS) components. In this chapter the evaluation of ready-made OSS components is discussed. Usage of OSS components in commercial software is an alternative choice to developing a complete software solution from scratch. Many components in software solutions are generic ones that are not specific to a given solution, hence considering a reuse of existing components instead of developing them is a valid option for constructing a software solution. However, a chosen component can affect the whole software solution quality in the same way as a design decision can. The difference is that in the case of ready-made components the design decisions are part of the component. Therefore, designers choosing specific components to be used in a software solution should have tools that allow them to assess the suitability of a component in a given context.

The choice of this quality improvement area was driven by the need of improving and systematising the evaluation process in the case organisation. The goal of the improvement was to make the evaluation more efficient and take risks of component usage into account. Again the need for the improvement was observed internally and fulfilled by assembling the evaluation framework.

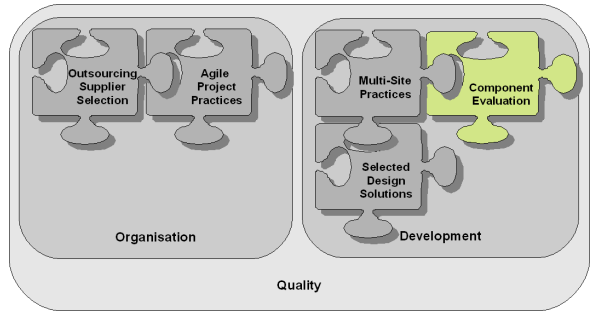



**Figure 5.1** Component evaluation

In this chapter the basic concepts of Software Product Lines (SPL) and main OSS evaluation methods are presented first in Section 5.1. The understanding of SPL and main OSS evaluation methods is needed as they are conceptually merged together

in the OSS component evaluation framework. The actual framework is presented in Section 5.2. Finally, this chapter is summarised in Section 5.3 and presented contributions are discussed in the context of the research questions of this thesis.

## 5.1 Relating OSS Evaluation to SPL

The OSS component evaluation framework that is presented in Section 5.2 is the main contribution in this chapter. Moreover, the framework was put into the context of SPL construction, which is a new conceptual point of view for OSS evaluation. Therefore, we need to briefly discuss the context of framework usage and both concepts, SPL and main OSS evaluation methods. Details of considering OSS component evaluation as an SPL construction process have been presented in publication [P3].

To understand the reasons why the OSS component evaluation framework was constructed in this particular way, we should discuss the contexts for its usage. The framework combines known good practices of OSS evaluation with concepts known from SPL. The framework is based on many criteria selected based on feasibility in commercial settings. A component in the context of this evaluation framework can refer to any software part that can be used in a software solution. For instance, a component can be a simple Java library which is dynamically used in a solution, a whole framework, or a full system that is customised to particular solution needs.

Additionally, the framework is meant to be used in a specific environment of a software service company, which develops customised software solutions for specific problems of their clients. Even though the solutions differ to a large extent at a detailed level, to some extent common elements among them can be found. In this context OSS components can be often reused in many solutions that share some portions of common functionality. Evaluation of such components must take the context in which the component is used into account and assess any risks associated with the component's usage. For instance, a solution being developed can be very complex and risky by its nature, and therefore any additional risk coming from a new component can be jeopardising for the whole solution, which should be mitigated by a proper component evaluation. In addition to the solution complexity also the component role makes a difference. A component that is a key component in the whole solution should be scrutinised much more carefully than a component that plays only a supporting role. Therefore, the effort and level of detail of the OSS component evaluation should be adjusted to the solution context, and evaluation tools should be adjusted to the needs of the evaluation process.

In the following sections existing OSS evaluation methods are presented first in Section 5.1.1. Next, basic SPL concepts are presented in Section 5.1.2.

### 5.1.1 Open Source Component Evaluation Approaches

Many tools have been developed just for evaluation of OSS. These tools can be successfully used as such, modified, or even some evaluation results of well-known software can be found. A comprehensive overview of OSS evaluation methods has

been presented in an article by David A. Wheeler [134]. In addition to collecting references to existing evaluation methods, Wheeler proposes a method for comparing OSS components. The method consists of four steps: *identify candidates, read existing reviews, briefly compare the leading program's attributes to your needs, and then perform an in-depth analysis of the top candidates* [134]. This method is meant for OSS comparison. However, in the case of evaluation of OSS for reuse in software solutions discussed in this chapter there can often be one candidate that should be evaluated for its quality and for possible risks introduced to the solution. The functionality evaluation is naturally important as well. However, some components can be evaluated for their potential use in future software solutions without explicit requirements for functionality. In such cases the functionality evaluation can be limited to recording the provided functionality. Despite a different purpose, the evaluation method presented by Wheeler includes some criteria that can be applicable in any context. For instance, the method suggests criteria like support, security, or legal/license issues, among other criteria [134]. Those criteria are also used in the context of OSS component evaluation for commercial software solutions, as proposed in this thesis in Section 5.2.

Other prominent evaluation methods include, for example, Qualification and Selection of Open Source software (QSOS) [7, 114], Open Source Maturity Models by Navica [90] and Capgemini [23], or Business Readiness Rating (BRR) [20]. QSOS [7, 114] consists of four steps:

1 definition, which specifies the scope and context of the evaluation,

2 evaluation, which covers functionality and risks for users and service providers,

3 qualification, weighting the criteria according to an axis specified in evaluation, and finally

4 selection, which can be strict or loose depending on needs.

The QSOS method is based on a system of weights and point scores that can be adjusted depending on the needs. Additionally, a QSOS method is under GNU license itself including evaluation reports that are available at their website [1] and can be a valuable source of information for component evaluation.

Another distinctive evaluation method is Business Readiness Rating (BRR) [20], which aims at finding out the suitability of software for usage in a commercial context. BRR proposes a set of criteria that can be adjusted depending on requirements, as well as templates using different weights for various metrics and score points. Similarly to QSOS BRR offers ready templates for certain OSS on their website [2], which can be a helpful starting point for evaluation. Other methods, namely Open Source Maturity Models by Navica [90] and Capgemini [23], provide their own contribution to OSS evaluation. For instance, Capgemini's Open Source Maturity Model (OSMM) [23] uses four criteria types and 12 criteria to assess maturity, compare, and select OSS. Interestingly, however, some researches found that formal evaluation methods are not widely used in industry, at least such an observation was reported

[1] http://www.qsos.org

[2] http://www.openbrr.org/wiki/index.php/Downloads

in the Norwegian cases reported by Hauge *et al.* [57]. This report has not been discussed in publication [P3]. However, it provides an additional and recent account of OSS evaluation in commercial software companies. Hauge *et al.* report that in the investigated cases OSS evaluation was rather informal, and often an important role was played by recommendations from a social network or reviews found on the Internet [57, p. 44].

Generally, all of the above-mentioned OSS evaluation methods provide a valuable source of possible criteria to be used for evaluation. Moreover, they provide in many cases ready-made evaluation reports. Some criteria used in these evaluation frameworks were also used in the criteria for OSS component evaluation proposed in this thesis, as listed in Section 5.2. Additionally, the OSS component evaluation framework advises to use any existing evaluation results, such as ready-made evaluation reports, as a source of information. The main difference between the official evaluation frameworks and the proposed OSS component evaluation framework (see Section 5.2) is the fact that the latter uses a specific set of criteria that was adjusted to a particular context. Additionally, the above-mentioned OSS evaluation methods use various point systems for evaluating individual criteria, while the OSS evaluation framework proposed in this chapter does not use any points for criteria evaluation and relies on the experience and judgement of the evaluator.

At a general level the evaluation process of an OSS component can be abstracted as depicted in Figure 5.2. The process typically starts from *requirements gathering*, and then *selection* of relevant software is done. Next, *functional evaluation* and *non-functional evaluation* are performed. Finally, a chosen component can be *integrated* with the whole system. These steps are quite general. However, they set the stage for combining typical OSS evaluation and certain aspects of SPL, which are presented in the subsequent section.

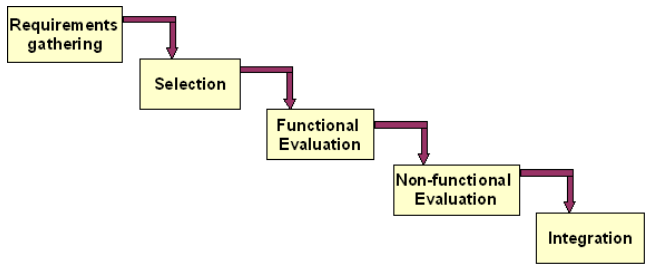



**Figure 5.2** A typical evaluation process [P3, p. 12]

### 5.1.2 Software Product Lines - Overview

Software Product Lines (SPL) have been created to ease the development of products that share common architecture and functionality. Paul Clements and Linda Northrop define SPL as follows: *A software product line is a set of software-intensive systems sharing a common, managed, set of features that satisfy the specific needs of a par-*

*ticular market segment or mission and that are developed from a common set of core assets in a prescribed way* [27, p. 5]. This definition indicated the important parts of any SPL, namely, the shared and managed set of features and the development based on common assets and in a prescribed way. The same requirements, i.e., a set of shared features and development based on existing assets, can be applied to OSS components meant for frequent reuse in multiple software solutions. Therefore, we should be able to look, to some degree, at the OSS evaluation in a similar way as construction of an SPL and building an asset set for the SPL.

There is much literature available on SPL development. For example, Linda Northrop presented a report on adoption of SPL in organisations based on specific patterns [92]. The report presents the Factory pattern [92, p. 5] that facilitates SPL adoption in organisations. Practical approaches to SPL development were presented in SEI's framework for SPL practices by Northrop *et al.* [93], which defines three main activities for SPL development. The SEI's framework includes *Core Asset Development*, *Product Development*, and *Management* as main activities for SPL.

In addition to patterns and practices for SPL construction, approaches to handling quality concerns in SPL have been defined. Niemelä and Immonen have specified Quality Requirements of a software Family (QRF) method that handles quality requirements in SPL [91]. The QRF method consists of five steps and engages main stakeholders in order to get a complete scope of the various quality requirements. Finally, a model for building SPL based on OSS components was also proposed, which has been presented by Ahmed *et al.* [3].

As can be seen, construction of SPL requires much consideration about the requirements, functional and non-functional ones, of the product family that the SPL is used for. The general steps of construction of an SPL have been defined by Jan Bosch [19, p. 189-190] as follows:

1 Business case analysis - which focuses on feasibility from the business point of view of the product development with SPL,

2 Scoping - which selects products and features that should be developed in SPL,

3 Product and feature planning - which considers possible features coming in the products developed in SPL,

4 Design of the product-line architecture - which specifies the SPL architecture based on the requirements gathered in the previous steps,

5 Component requirement specification - which specifies components used in the SPL architecture and the functionality of specific components and their usage in products, and

6 Validation - which validates that all requirements planned for SPL are correctly supported in the SPL.

Having the basics of SPL covered, it is now possible to combine the OSS evaluation with some SPL activities as presented next.

## 5.2 How to Evaluate OSS Component for Commercial Use?

OSS component evaluation for reuse in possibly multiple software solutions can be viewed similarly as SPL construction. Naturally, it is not the same process, first of all because the solutions are much more customised than products. However, the requirements for quality of components and possible risks related to their usage are similar to SPL. Secondly, multiple solutions are not governed, at least currently at Solita, in the same way as SPL architecture is. Therefore, instead of referring to fully fledged SPL based on OSS components, the software solutions are referred to as software Solution Lines [P3, p. 13]. Solution Lines refer to systems that comprise a custom solution that can be based on reusable components, but does not follow all component management procedures of SPL.

The very high level mapping between the main OSS evaluation steps and SPL construction steps is depicted in Figure 5.3. The evaluation steps concentrate on the components rather than on SPL as such. The first phase *Business case analysis* corresponds to *Requirement gathering* and *Selection*. This first evaluation phase should determine whether the OSS component should be considered for reuse, for instance, the license can be not suitable for usage in commercial solutions. Additionally, the business analysis should set the associated risk level and consequently needed evaluation effort. Next, *Scoping* is done, which corresponds to *Functional Evaluation*. The scoping for a general reuse case, which does not have any particular feature requirements, simply lists the features supported by the component. The third step is *Commonality and variability*, which combines SPL construction steps of product and feature planning and product line architecture, and which roughly corresponds to *Non-functional Evaluation*. At this stage the component's extensibility, for example, should be evaluated. Finally, the last stage of OSS evaluation in the context of SPL is *Design and validation*, which is seen as *Integration* in OSS evaluation. This last step validates the component by using it in a specific software solution. Even though this mapping between OSS evaluation and SPL is not perfect, it adds a perspective of high reuse systems to OSS component evaluation.

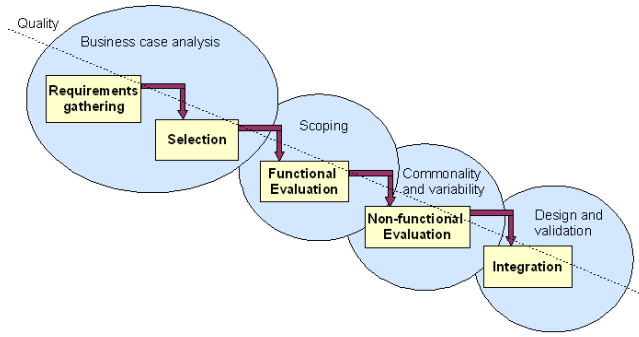



**Figure 5.3** OSS evaluation in SPL context [P3, p. 13]

### 5.2.1 Evaluation Criteria

The OSS evaluation framework described in publication [P3] is based on a number of criteria. The criteria were selected from a long list of possible evaluation criteria. Then those criteria were categorised in to groups for clarification. Finally, all the criteria were analysed as for their feasibility usage. Criteria that were overlapping with others were eliminated. Furthermore, criteria that were not easy to obtain and did not bring much value to the evaluation were removed. Finally, a consensus list with recommended criteria was created. These criteria were the suggested ones and they might not be applicable in all possible cases. The criteria summary with all categories is gathered in Table 5.1, and the details of all the criteria can be found in [P3, p. 13-16]. There were 8 different criteria categories investigating various aspects of OSS component. They were also relying on multiple sources of information, naturally using information from the OSS community, but the sources also included any other independent users, and internal experience in the organisation with the component. One particularly important category was the *No Excuses* category. The criteria in that category must have been fulfilled, otherwise the component evaluation would have been stopped. For instance, a non-standard license or one that prohibits usage in any commercial context would result in component rejection. The importance of other criteria categories depended on the context.

### 5.2.2 Framework Usage

The criteria list is intended to be used for an evaluation performed by an experienced developer. The fact that the evaluation is meant to be done by an experienced person is important, as the criteria are not evaluated using any weight and point system. The evaluation should encourage a critical consideration of the evaluator who makes the final recommendation as well as possible comments on each of the criteria. That summary can then be later reexamined if the already-evaluated component is considered for usage in a different solution.

The framework is part of the evaluation process used at Solita and it is available in the company's intranet. There an evaluation template as a wiki page was created. It contains additional explanation of the criteria and the expected evaluation steps, which can also cover project-specific evaluations that are not covered by this OSS evaluation framework. Results of evaluations are available for the evaluated components together with evaluation summaries.

The framework has been used for evaluating different workflow engines. That evaluation has been done and it is part of a Master's Thesis by Ossi Syd [122]. Three open source workflow engines were evaluated. Only one was selected as advised for usage in software solutions that require workflow engines. In addition to OSS evaluation, that evaluation case resulted in feedback on the framework and its usage. The feedback received from the evaluator included a comment that the framework was meant for an experienced evaluator and that the framework left many criteria for the interpretation and judgement of the evaluator. That comment was in line with the intentions of this framework. The feedback also included a positive note on the fact that the proposed framework did not use any point system. Finally, the evaluator,

**Table 5.1** Summary of possible evaluation criteria [P5, p. 16]

| Criteria Category | Criteria |
|---|---|
| 'No Excuses' | *Working copy*, *License*, and *Activeness* |
| Source Code | *Review*, *Metrics and Static Analysis*, *Dependencies* |
| | *Revision Control Comments*, and *Testing* |
| Documentation | *Instructions, Help*, *Cookbooks, Tutorials, How-tos*, |
| and Support | *API*, *Books*, *Training*, *Bug Tracking System*, |
| | *Number of Faults in the Bug Tracking System*, |
| | *Number of Open Faults in the Bug Tracking System*, |
| | *Bug Fixing Velocity*, and *Possibility of Contribution* |
| Community | *Active Forums*, *Promptness of Forum Responses*, |
| | *Unanswered Questions in Forums*, *Website Updates*, |
| | *Last Delivered Fix*, *Frequency of Updates*, |
| | *Number of Developers*, and *Developers' Motivation* |
| Other Users, | *In-House Users*, *Commercial References*, |
| Popularity | *Social Media*, *Other Publications*, *Number of Downloads*, |
| | *Exploit Reports*, and *Search Engines* |
| Maturity | *Version Number*, *Project Age*, *Version History*, |
| | *Localised Versions*, *Supported Platforms*, |
| | *Test Coverage*, and *Roadmap* |
| Future | *Extensibility*, *Related Standards*, |
| | *Unfinished Related Standards*, *Commercial Versions* |
| | *Can the Component Become Commercial?*, |
| | *OSS Project Model*, and *Effort to Maintain* |
| Legal Concerns | *Pending patent* and *Law suit/charges against* |

who used the framework for the workflow evaluation, was positive about the fact that the framework focused on OSS component quality. The quality focus provided additional and helpful information for the evaluation. In addition to the workflow engines evaluation, the framework has been used at least once. Based on the positive feedback received, the framework is now part of Solita's general evaluation process. Therefore, the framework is likely to be used also in the future. Finally, as with many processes, it is likely to be further improved in the future to adjust it to new needs.

## 5.3 Summary

This chapter presented an evaluation framework for Open Source Software components for reuse in software solutions. The component reuse can be seen as a com-

plementary part of specific design solutions, which is discussed next in Chapter 6, and development of software from scratch. The presented framework puts a strong emphasis on quality of OSS components and possible risks attached to component usage in commercial software solutions. The importance of quality aspects and reuse are mapped to the main steps of Software Product Lines creation. Even though the mapping is not direct, there is a certain degree of similarity in the requirements for components used in SPL and software Solution Lines. The OSS evaluation framework takes into account a number of criteria, which are adjusted to a commercial context of evaluation, from different sources. Moreover, the framework encourages critical thinking of the evaluator and recording of the decision process by using criteria without any weight or point system, and instead uses descriptive criteria summaries. The framework is currently in use as part of an evaluation process at Solita.

The framework guides the evaluator in the evaluation process by providing various criteria groups, which cover different aspects of the OSS component that may influence the risks related to using the component. Also, the framework defines the role of the evaluator, who is expected to use critically the framework and criteria. An organisation using the framework is able to assess the quality of an OSS component and possible risks attached to the component, which in turn affects the whole software solution where the component is to be used. Therefore, this framework provides a way for a software organisation to improve the quality of its software by providing a tool that can assess quality of individual OSS components. This contribution addresses the research question RQ1 in this area of component evaluation. A limitation in the scope of this improvement is the extent to which it has been used and consequently available data that could be used for drawing wider conclusions. However, in the *Development* category of improvements the improvements directly affect the created software. In the case of OSS component reuse the impact depends on the degree to which reusable OSS components are utilised in a software solution.

The framework helps in component selection in a way that mitigates the risks related to the component usage in a commercial software solution. Additionally, the knowledge base about OSS components build from evaluation results can be used in the organisation as a reference material when components are considered for use in new solutions. The framework is another example how a software organisation can internally define a process that addresses needs in a specific area of quality improvement. The framework was developed based on observed needs and is currently a part of the internal evaluation process, which partially answers the research question RQ2.

Furthermore, OSS component evaluation can be related to specific process improvement areas in the CMMI for Development framework [124]. The evaluation of OSS components as part of component reuse can be seen as part of the Technical Solution (TS) process improvement area in CMMI Development [124, p. 456]. The decision to use a specific component is also a design decision. Therefore, quality improvements in the component selection process seem to belong to the TS process area, the goals of which are listed in Section 6.4.

Additionally, the evaluation eliminates risks related to component usage that potentially might not be eliminated. Therefore, component evaluation can be seen as part of the CMMI Development Risk Management (RSKM) process improvement

area [124, p. 420]. The purpose of the RSKM process area has been defined as *'to identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives'* [124, p. 420]. The risk management related to usage of OSS components in a project and identification of the risks by evaluating the component can be seen as part of a specific practice within the RSKM process area. The OSS evaluation suits best the RSKM practice SP 2.1 Identify Risks [124, p. 425]. Naturally, neither TS nor RSKM process areas are fulfilled by the presented OSS evaluation framework. However, the aim of the framework is in line with a subset of goals of these two process areas of a major quality improvement framework. Therefore, it is possible to state that an organisation that decides to use such an evaluation framework for component selection partially follow recommendations of an official quality improvement framework, which answers research question RQ3 within the scope of this area of component selection.

# CHAPTER 6

# Development - Selected Design Solutions

Selected design solutions are the fifth and last area of quality improvements that this thesis addresses and makes contributions to (see Figure 6.1). Design decisions are made by individual designers in a software company. Therefore, knowledge about the consequences of using specific design solutions is relevant for relatively many people in a software organisation.

Design of software systems and their components is an integral part of software creation in companies developing software. Even small design decisions may have effects on entire systems, which consequently may be noticeable to the final user. Therefore, it is important to make design decisions consciously. In this chapter the impact of selected design solutions on performance is presented, based on empirical data. Additionally, the empirical findings are used to demonstrate how such findings can be used in tools that support quality-driven design.

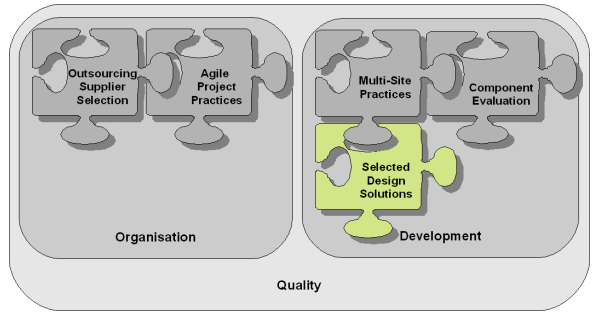



**Figure 6.1** Selected design solutions

In order to understand the selected design solutions, we should start ( Section 6.1) by discussing general approaches to software design, tool support for software design, and their relation to software quality. Then (Section 6.2) we can discuss findings on the impact of selected design patterns on performance in distributed systems. Next, a tool support for quality-driven design is proposed in Section 6.3. Contributions of this chapter are summarised in relation to the main research questions of this thesis in Section 6.4.

## 6.1 Software Design Considerations

Design of software systems has been under research for a long time. One of the main works in this area is the catalogue of various design patters by Gamma *et al.* [46], who categorised and described in detail various design patterns. Designers use design patterns on a daily basis. This design pattern catalogue allows designers not only to have a reference point for commonly used solutions but also to have names for those solutions, which facilitates communication. The design pattern catalogue by Gamma *et al.* [46] is primarily relevant for any object-orientated programming languages. However, there are also other design patterns that are specific to certain technologies. Examples of design patters only for Java-related technology include the pattern catalogue for Java [1] by Mark Grand [53], the J2EE [2] design pattern catalogue by Alur *et al.* [5] and Enterprise Java Beans (EJB) patterns by Floyd Marinescu [84]. Naturally, there are also other patterns for different technologies. For example, Trowbridge *et al.* [126] presented patterns for .NET [3] technology, Thomas Erl presents Service Oriented Architecture (SOA) patterns [37], or Ajax design patterns by Michael Mahemoff [82], to mention only a few.

Even though the technology-specific patterns are closely coupled to their implementations, they still in many cases can be easily linked to the more generic patterns. For example, Session Facade [5, p. 291] is a J2EE-specific implementation of the classic Facade design pattern [46, p. 185]. Similarly Command design in Java [53, p. 277] can be easily linked with the generic Command design pattern [46, p. 233].

Additionally, there are patterns that are meant for specific usage and are not necessary bound to any particular technology. Architecture patters, for example, provide examples and systematise solutions to common problems in software architecture. Architecture patterns have been presented, for instance, by Buschmann *et al.* [22] and Martin Fowler [44]. Other examples of usage-specific patterns could include re-engineering patterns by Demayer *et al.* [30], applicable for existing software refactoring, or analysis patterns by Martin Fowler [43], applicable for problem conceptualisation at the early design stage. All these examples show the variety of design pattern types as well as applications for them. The design considerations presented in Chapter 6 are limited only to a small sub-set of design patterns. Moreover, the design patterns discussed in Chapter 6 have been implemented in two technologies, namely J2EE and .NET.

Another aspect of design is the relation of design decisions to specific quality attributes. According to the IEEE definition a quality attribute is '*A characteristic of software, or a generic term applying to quality factors, quality subfactors, or metric values*' [115, p. 3] and a software quality metric is '*A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality*' [115, p. 3]. Various metrics for assessing software quality and related design have been proposed. For example, Stephen Kan collected a large number of different software quality-related metrics, including metrics measuring design complexity for object-oriented software [67, p.

[1] `http://java.sun.com/javase`
[2] `http://java.sun.com/j2ee`
[3] `http://msdn.microsoft.com/netframework`

334]. In addition to complexity metrics, software metrics were proposed that were optimised for design pattern application in refactoring, as reported by Muraki and Saeki [89]. Quality attributes have been categorised already for a long time. For example, Barbacci *et al.* [11] discuss selected quality attributes, namely performance, dependability, security, and safety, and relations between those attributes. Additionally, Barbacci *et al.* discuss different trade-offs between the attributes, and they show that an optimal fulfilment of one attribute happens at the expense of another one [11, p. 4]. The trade-offs between various quality attributes and their relation to design decisions have been further researched by Bass *et al.* [13,14] and Bachmann *et al.* [9,10].

Finally, tools supporting the design process are not a new concept. Already Potts and Bruns [101, p. 419] note a need for documenting both the process, for design decision reasoning, and design results. They present a model documenting the design process or deliberation [101]. They also indicate a need for a tool that would support the designer in design process documentation. At the architecture design level a '4+1' view model has been proposed by Philippe Kruchten [76]. This model uses four different views on system architecture (i.e., logical, process, physical, and development view) and use cases that span across the views. Additionally, Kruchten lists tool examples for the different views. Another tool that supports a subset of the different architectural views has been presented by Rick Kazman [71]. Moreover, Kazman specified seven generic requirements for an architectural and design tool, most of which are fulfilled by the actual tool SAAMtool.

## 6.2 How Can Selected Design Solutions Impact Performance in Distributed Systems?

In order to demonstrate the impact of selected design choices in distributed systems, we start from analysing results of empirical case studies performed for two popular technologies used in web applications. The details of the cases can be found in publications [P8] and [P6]. The objective of this study was to find out whether design choices that do not affect an application's functionality can affect a specific quality attribute, namely performance. The chosen design patterns were Facade [46, p. 185], Command [46, p. 233], and Combined Command, which was an implementation of Macro Command [46, p. 235]. These design patterns were selected as they all are valid choices for implementation of an interface to an application's business logic that can be accessed remotely. The selected design patterns were implemented in two technologies popular in distributed applications, namely J2EE and .NET. In J2EE technology the design patterns were implemented as Session Facade [84, p. 5], EJB Command [84, p. 18], and a modified EJB Command that handled multiple commands at a time, as in Macro Command [46, p. 235]. For .NET technology the patterns were ported to C# technology and exposed for remote access through .NET remoting [94,103].

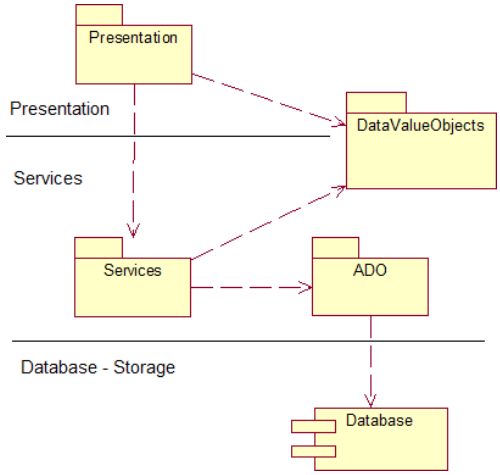



**Figure 6.2** General DVC architecture overview [P6, p. 2]

### 6.2.1 Study Setup

This study was conducted as two cases that were technology-specific. First, the results for J2EE technology were collected and analysed. They are described in detail in publication [P8]. Then the same case was ported from J2EE technology to .NET technology and the same tests were conducted. The results are described in publication [P6]. Furthermore, in publication [P6] general conclusions were drawn based on the two cases. The following sections summarise the findings.

In both cases the same test application was used. The application was a simple Document Version Control (DVC) system that allowed for storing documents, versioning them and access restrictions for users. The application was implemented as a layered application [22, p. 31] with presentation layer, business logic layer, and persistence layer in MySQL database [4], as depicted in Figure 6.2. The presentation and business logic layers were respectively implemented as Java Server Pages (JSP) and Enterprise Java Beans (EJB) for J2EE technology, and ASP pages and C# code accessed through .NET remoting [94, 103] in the case of .NET technology. The three design pattern variants implemented to provide specific functionality in the DVC application are presented in Figure 6.3. The patterns were used in the business layer to expose functionality to the presentation layer. The design patterns were implemented so that services (methods) exposed by Facade corresponded to separate Command pattern implementations. The Combined Command allowed for execution of a few commands in one call. That possibility was used to combine a few commands together that otherwise would be executed as separate calls. The patterns did not impose any restrictions in this case for the provided functionality, and therefore they could be treated as design alternatives for this application.

The performance of the application was measured based on selected metrics that

[4] `http://www.mysql.com`

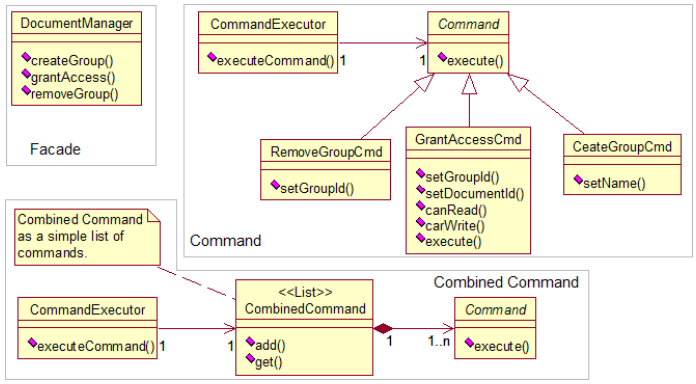



**Figure 6.3** Example pattern implementation [P6, p. 2]

consisted of

1 throughput - measured in requests per second,

2 average response time - measured as the average time that was needed to obtain a response by the client,

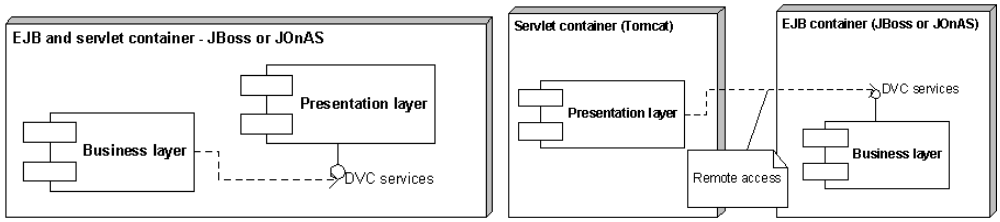
3 threshold exceeded - measured as a number of requests whose processing time took more than a threshold of 10 seconds, and

4 success rate - measured as percentage of successfully completed requests of the total number of requests.

The performance-related metrics were measured using the JMeter [5] tool that simulated requests executed according to the provided scenario for the given number of simultaneous users. The test scenario included a few activities for each user. The user activities generated calls from the presentation layer to the business layer where the investigated design patterns were implemented. The number of users was increased gradually from five to 220, which was determined empirically as the maximum number of users that can be handled by the application. The tests were repeated in four series in order to obtain reliable data. The application in all the cases handled data consisting of 500 users and 1500 documents.

The DVC application was deployed in technology-specific deployment variants. For the J2EE technology case two deployment configurations were used. One configuration involved only one J2EE container for the presentation and business logic layers, which was called *local deployment* and can be seen in Figure 6.4(a). The other deployment configuration consisted of two separate containers, one for the presentation layer and the other one for the business logic. That separate arrangement of containers was called *separate deployment* and can be seen in Figure 6.4(b). The local deployment allowed for container-specific calls optimisation, while separate de-

[5] `http://jakarta.apache.org/jmeter`




(a) Local deployment - one container.

(b) Separate deployments - two containers.

**Figure 6.4** Deployment variants [P8, p. 14]

ployment forced remote calls (i.e. Remote Method Invocation (RMI) [6]) between the containers. The EJB containers used were JBoss [7] and JOnAS [8]. The Tomcat [9] servlet container was used for the presentation layer in the case of separate deployments.

In the case of .NET implementation of DVC, the application was deployed in Internet Information Services (IIS) [10]. The .NET case deployments were done in similar configurations as in the J2EE case with one local deployment configuration where presentation and business logic layers were deployed in the same container. The second .NET deployment case involved two containers on separate nodes, one for presentation and the other one for business logic layers. In addition to the deployment variants in the .NET case, different configurations of the remote communication used between the layers were used. Two transport layer variants were used: one used HTTP protocol for communication, the other one used 'raw' TCP communication. Finally, for the HTTP communication case two different serialisers that convert data transferred between the layers were used. The serialisers used were an XML serialiser and a binary serialiser. All these configuration variants were investigated in order to find out possible differences between performance of the application when different design patterns are used for the inter-layer communication. Having presented the general setup of the cases we can next discuss the main findings of each investigated case.

### 6.2.2 Findings in J2EE

In the case of local deployment the results were not very clear and had much 'noise', which was visible in the performance metrics graphs as unstable and uncorrelated lines (see Figure 6.5). The throughput values for the investigated patterns do not differ noticeably between each other. What could be observed was that the throughput of the application lowered with an increasing load, which is an expected be-

[6] http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html
[7] http://www.jboss.org
[8] http://jonas.objectweb.org
[9] http://jakarta.apache.org/tomcat
[10] http://support.microsoft.com/ph/2097

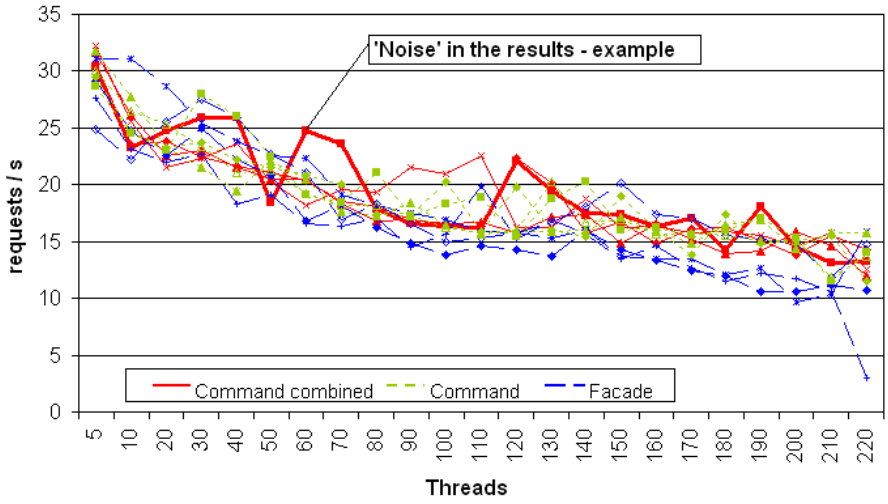



**Figure 6.5** Throughput - J2EE local deployment [P8, p. 17]

haviour. Furthermore, it was possible to notice that the obtained throughput values were higher compared to separate deployment, which is depicted in Figure 6.6. This observation was also expected as in the case of separate deployment remote communication between containers causes additional overhead.

The results obtained from tests in the case of separate deployment were most interesting as they enforced remote communication between containers. The throughput results can be seen in Figure 6.6. There are clear differences between the pattern used for accessing the business logic in the DVC application. Facade results show that in this case simple calls to Facade's service methods were most efficient. Then Command combined was about 25% slower than Facade, however about 20% faster than Command. In this case calls to Facade did not involve any overhead provided by the data structure as only the method parameters were passed. In the case of Command the same parameters as in the case were passed and additionally the Command object itself had to be transferred. In the case of Command combined the data complexity was similar to the Command case. However, Command combined provided a possibility of executing a few commands in one call, which reduced the overhead of multiple remote calls.

Other metrics measured showed the same results, especially the average response times. The success rate showed when the container was not able to compare with the load, which happened approximately when the load reached 160 threads, as well as the percentage of correctly served requests. The success rate additionally showed when the waiting time for response increases with load. Finally, the differences between different containers were noticed. There were differences in values obtained as well as the stability of each container. However, a container performance measurement was not an objective of this case, and therefore the results are not analysed from that point of view. What was important was the fact that the noticeable differ-

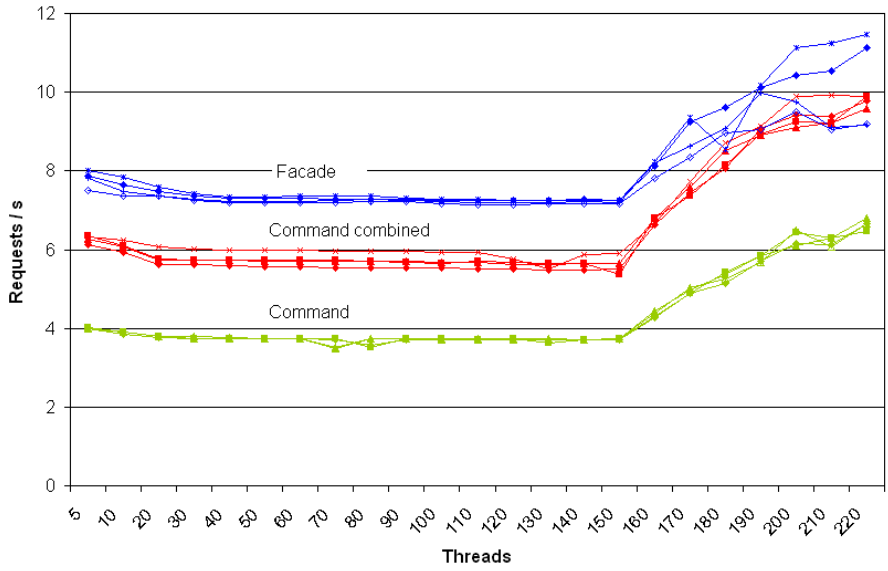



**Figure 6.6** Throughput - J2EE separate deployments [P8, p. 18]

ences between patterns had similar character regardless of the container.

### 6.2.3 Findings in .NET

Similarly as in the case of J2EE, in the local deployment case of .NET no clear differences between the investigated design patterns were observed [P6]. There were no significant differences between the formatters used either. It should be noted, that the communication in the case of application deployed to IIS container included serialisation. However, there was no network communication as the deployment was local.

When the DVC application was deployed on separate machines, differences in performance were observed. For example, the throughput diagram for separate deployment can be seen in Figure 6.7. It can be seen that generally Command had the worst results. Facade and Command combined had similar results, which differed from each other only in specific cases.

Average response times showed results in line with what the throughput results indicated. Success rate and the exceeded threshold pointed out the number of users with which the application started to have problems with serving requests, which happened with 20 or more users. Additionally, the results for binary and SOAP formatters both showed results that confirmed the differences between the patterns.

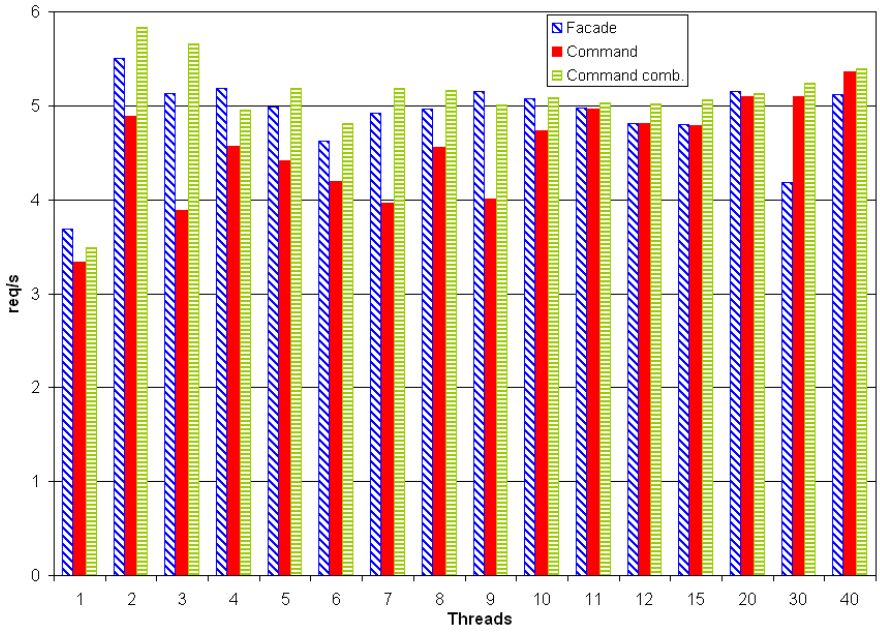



**Figure 6.7** Throughput - .NET separate deployment [P6, p. 8]

### 6.2.4 Serialisation Considerations

As could be observed in both technologies, J2EE and .NET, the most significant differences were observed when communication between the application layers was performed remotely. Remote communication in generally includes data serialisation to a format determined by the technology and formatter used, and next sending the data over the network to the other layer where it would be deserialised and served in the other layer. These operations seemed to have the biggest impact on the performance and in that context revealed differences between the design patterns. Therefore, the remote communication was investigated in detail in order to obtain more data and find out the reasons for the differences. The serialisation and remote communication tests included [P6, p. 8]:

- Java RMI,
- .NET Remoting with SOAP formatter over HTTP,
- .NET Remoting with binary formatter over HTTP,
- .NET Remoting with binary formatter over TCP, and
- .NET Remoting with SOAP formatter over TCP.

The measurement of size of the data for each design pattern revealed the impact of the patterns on the amount of data that has to be transferred in each case. Regardless

of the formatter used, Facade had the smallest data size, then was Command, and the biggest data amount was for Command combined. The differences for small objects serialised were most significant. However, for the largest objects, this difference was not so significant. Furthermore, there were naturally differences between formatters, too. The SOAP formatter produced the largest data, which was a few times larger than RMI or .NET binary formatters produced.

### 6.2.5 General Findings

The overall findings showed that in locally deployed application layers the choice of a design pattern used for communication between layers does not play a significant role in the performance of the whole application. This is due to the fact that in the case of local deployment there is no real over-the-network communication. However, when the layers are deployed separately there are noticeable differences in performance of the application depending on the design pattern used. For both J2EE and .NET, the Facade design pattern had the best performance among the three investigated ones. Then the Command design pattern had better or similar performance as the Command combined pattern for single calls. Additionally, based on the serialisation investigation of the patterns it was known that the differences in performance come from the fact that each of the design patterns introduced a different degree of additional complexity to the remote call. Facade required only transfer of the parameters used in the call, while Command and Command combined had additional structure of the Command itself to be transferred. It was also observed that the performance penalty caused by the design pattern used played a less significant role when the data transferred was large. Furthermore, the choice of the formatter used was important as the data size produced by each formatter differed, which was especially visible in the case of the SOAP formatter. Finally, when remote calls could be optimised to one call that used Command combined that pattern had the potential to reduce the overhead of multiple remote calls and outperform at least multiple Command calls and be close to the results obtained by Facade.

Generally, the results showed how selected design decisions can affect the performance of a whole application. Designers can utilise this knowledge in their everyday work. From the viewpoint of organisations it is important to acquire thorough understanding of design decisions in the specific context of application architecture and technology used.

## 6.3 How Can Tools Support Quality-Driven Design?

The empirical findings on the impact of selected design patterns on performance of distributed systems, which were summarised in Section 6.2, can be utilised by tools supporting software design. The tool support for quality-driven design was the aim of a solution created using the MADE [11] tool [55]. The MADE 'Modelling and Architecting Design Environment' tool was an older version of a tool that has evolved

[11] http://www.cs.tut.fi/~mda/

into the tool currently known as Inari [12]. MADE is a design tool that uses a system of patterns for design descriptions. A MADE pattern consists of roles, which can be bound to, e.g., classes in a UML diagram, or other system artifacts, that specify relationships between artifacts bound to the roles [54, p. 144]. Additionally, MADE generates a list of tasks that have to be done in order to implement a pattern. The tasks represent unbound roles, i.e., roles that do not have concrete artifacts bound to them. A designer can implement a pattern based on the task list by binding particular roles to various artifacts, e.g., classes [54, p. 146].

An example overview of a MADE pattern and its mappings to a UML class diagram is presented in Figure 6.8. In this example two alternative solutions *A* and *B* are presented as a UML diagram. For the selected solution *B* the following MADE roles are selected: *AbstractB*, for the abstract representation of the solution, and *BImplementation*, for solution implementation representation. The example also lists tasks needed for implementation of the chosen solution. The tasks dynamically change depending on the solution selected and possibly missing bindings between roles and concrete artifacts. In this example, the task list includes tasks for implementing solution *B*. The *Provide abstract B* mandates the designer to link the role *AbstractB* with an abstract class. The second task mandates linking the role *BImplementation* with an implementation class *BImplementation*. When both tasks are completed, the MADE pattern is implemented. In this very simple example, the resulting solution is an implementation of an abstract class *AbstractB*. In the case of complex systems the tasks allow for gradual implementation of patterns and tracking of the implementation progress.

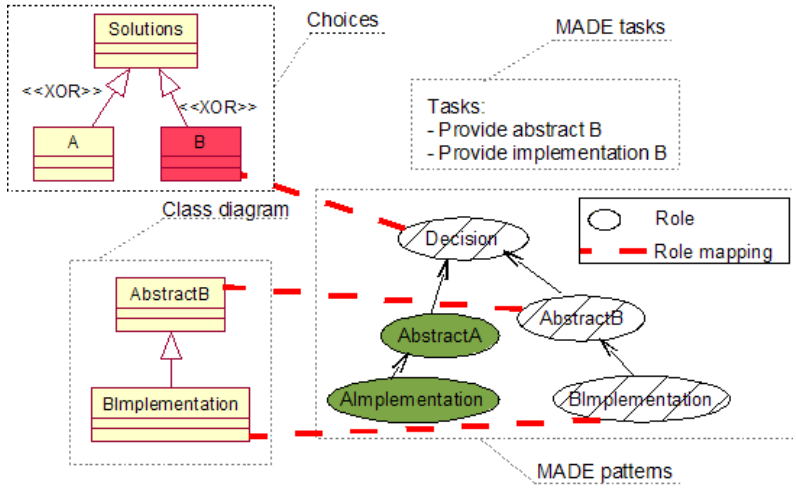



**Figure 6.8** MADE patterns and UML diagram example [P7, p. 197]

[12] http://practise.cs.tut.fi

### 6.3.1 Example Solution

The MADE tool was used for implementation of a solution to a specific design problem. This usage of the tool was an example of quality-driven design tool support, as the design choices were based on the impact the design has on specific quality attributes. The problem that was solved by the tool was an interface design for an example application. The example application was a Document Version Control (DVC) system already presented in Section 6.2.1. However, the application specifics were not of the main concern in this case. The problem was limited to the design of two interfaces for accessing the DVC application. One internal interface was supposed to provide a high performance access. The second, external, interface was meant as a highly flexible interface. In this context the interface designer was to design the interfaces based on functional requirements as well the non-functional ones.

Based on the findings on design pattern usage for remote access, which were presented in publications [P8,P6] and summarised in Section 6.2, a set of solutions were defined in MADE as a set of patterns. The solutions had a descriptive summary of their properties, so that the designer could make conscious choices between the solutions. Once a solution was selected, a list of tasks to perform was created. The list showed the minimum steps that had to be followed to implement the solution.

For the presented problem, three solutions were created in MADE. There was a solution based on the Facade design pattern [46, p. 185], the Command [46, p. 233] design pattern, and a Command combined [46, p. 235]. These three design alternatives were linked with specific quality attributes. The Facade-based solution was attributed high performance, but it was also characterised as the one with an inflexible interface. On the other hand, the Command-based solution was attributed a low performance among the provided solutions. However, the Command-based solution was also attributed a flexible interface. Finally, the Command combined-based solution was attributed average performance and interface flexibility the same as in the case of the Command-based solution. All this information about the solutions was available for the designer. Thus, based on the requirements and known quality attributes of the solutions, the designer could make a conscious choice between the solutions. Once the choice was made, the tool provided a list of tasks needed for actual implementation.

An example of a chosen solution for high performance is presented in Figure 6.9. It is a solution that fulfils the internal interface requirements for the DVC application. Once the designer selects the solution, the roles involved in it are activated. The roles are mapped to specific class roles in the UML class diagram. Finally, a list of tasks for the developer is created.

An example view of the MADE design environment is presented in Figure 6.10. The view consists of four parts, a pattern tree, marked as *Patterns*, a list of various roles, marked as *Role bindings*, tasks that need to be fulfilled to bind roles, marked as *Tasks*, and a selected task description window, marked as *Task description*. The pattern tree groups patterns and their implementations (instances). In the example one pattern *RemoteInterfaceSolutions* is visible and two instances that represent different implementations of each pattern. The implementation results in a UML class diagram that includes classes, or interfaces, mapped to roles of a MADE pattern. The role

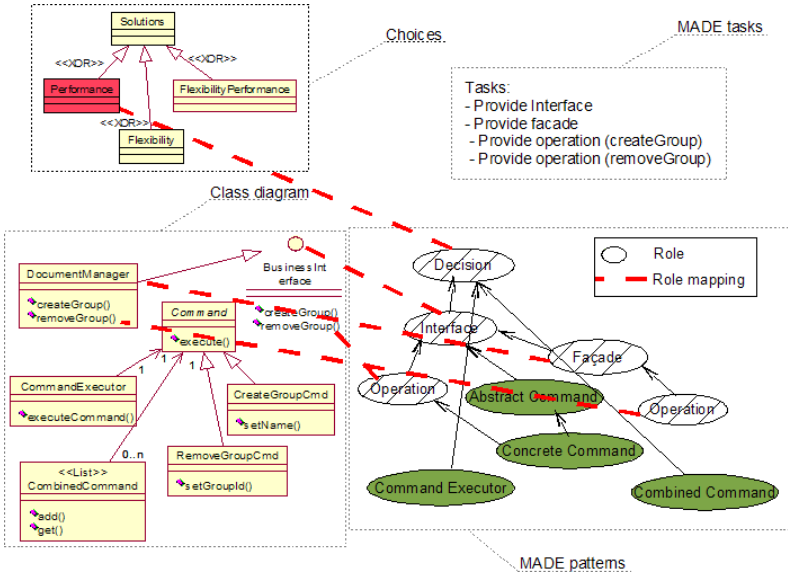



**Figure 6.9** MADE patterns and UML diagram for performance solution [P7, p. 200]

binding view shows roles that constitute a pattern, in this case the *RemoteInterfaceSolutions* pattern. The selected pattern has two types of roles: class roles and method roles. In the example, the *I_DocumentManager* interface is not bound to any implementation, which is marked in the view with a small circle in the bottom right hand side corner of the class role icon. For the missing bindings, the task list view lists tasks that have to be implemented, in this case the task is *Provide 'BusinessInterface'*. The meaning of the task is textually described in the task description view. In this example a flexible solution has been selected. It can be seen that a specific pattern has been selected and role bindings are present. Also one specific task is selected as well as its description. The pattern roles are bound to the UML class diagram managed in IBM's Rational Rose [13] (see Figure 6.11). The diagram shows a *Possible solution* corresponding to the design options that the designer can select from, a particular solution implementation marked as *Flexible solution implementation*, and a *Business interface* that corresponds to the role of *I_DocumentManager* from Figure 6.10. The link between MADE and Rational Rose is interactive, so that changes in the class diagram that break any bindings are visible in the MADE views and active tasks. This behaviour ensures that changes in the design do not break the already implemented functionality.

[13] urlhttp://www-01.ibm.com/software/rational/

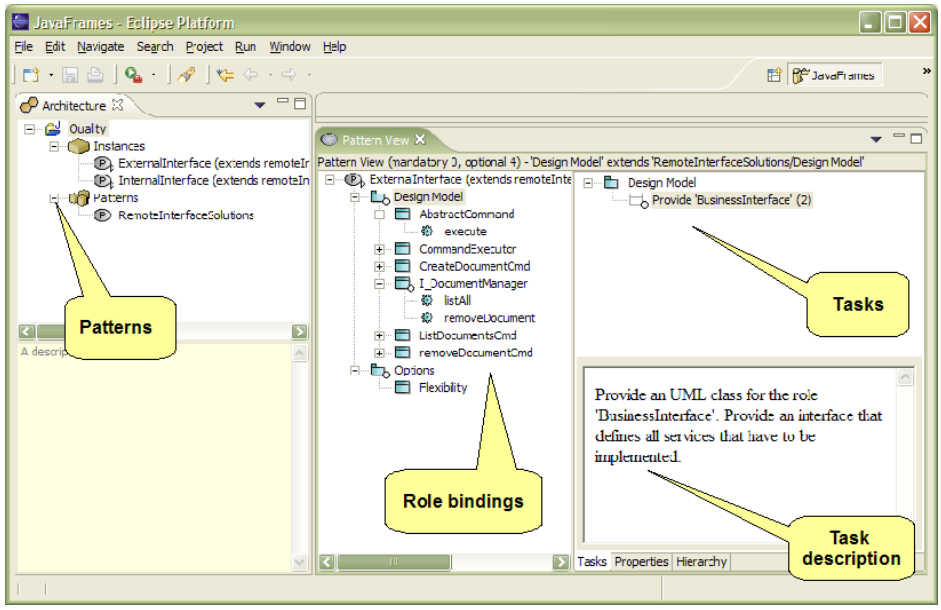



**Figure 6.10** MADE design view [P7, p. 203]

### 6.3.2 Tool Usage Benefits

Using MADE and the example set of solutions the designer was able to select a solution that was the most suitable to the given requirements. Naturally, the solution set in this example case was very limited and the presented problem was not very complex either. However, the aim of this case was to demonstrate that a tool can be used for quality-driven design. In the demonstrated case the experienced designer was likely to know the differences between design choices. However a less experienced one might find the tool useful. A tool that offers a wide range of design solutions together with guidance on the solution implementation could be even more useful.

This usage of MADE was only a proof of concept validation that showed potential applicability of such tools. However, the tool has never been used for any commercial purposes, mainly due to the fact that simple design choices are easily made by experienced designers. Additionally, a different tool set has been used at Solita. Despite this limitation, generally, the idea of a tool that helps a designer in making design decisions seemed useful. Organisations can build up a design library that is applicable for their use. The library can be used for development, but it can also be used as training material for new developers. The ways of utilising various tools in software organisations naturally depend on their needs, and in the case of MADE and the case company the practical applicability was found limited.

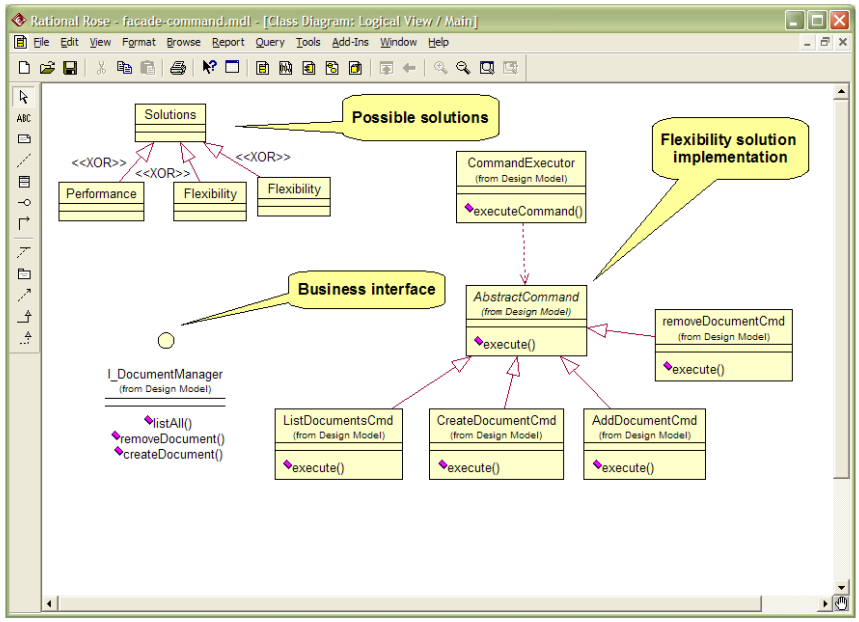



**Figure 6.11** UML class diagram in Rational Rose [P7, p. 203]

## 6.4 Summary

In this chapter we have presented how specific design choices can affect performance of a distributed system. The presented empirical results provide background information and justification on usage of specific design patterns for remote communication in distributed applications. Generally, the results also showed mechanisms that cause differences between the design patterns, i.e., data structure that results in increased size of serialised objects, as well as usage of particular technologies for serialisation, i.e., binary serialisation and XML serialisation. The data complexity increased the size of the payload to be sent between application layers, as did the XML serialisation compared with binary serialisation. Consequently, the increased size of data to be sent between application layers impacted the application performance.

This contribution is narrow but provides additional value to a designer far making conscious design decisions. Moreover, this chapter presented how results from an empirical investigation of specific design choices can be utilised in tools supporting design. The tool support for quality-driven design is both directly helpful in the design process, but also it allows to build up a knowledge base.

Software organisations can use the provided results directly. However, naturally they can themselves investigate, or use existing research on, other design solutions

in technologies and application scopes that are most important for their everyday design decisions. That way organisations are able to build up a knowledge base specific to their domain that can be utilised in the whole organisation. Therefore, the presented empirical results and tool usage contribute twofold by presenting how specific design solutions affect distributed systems and how the empirically gained knowledge can be utilised further in tools. In Solita the obtained results have been made available to everybody in the company's knowledge sharing tool, i.e., a wiki system.

In the context of the research questions of this thesis, this quality improvement area is important as it impacts the final software solution offered to end customers. Owing to the fact that any software system requires design decisions that impact the final software solution, this quality improvement area can have an impact on the overall quality improvement measures in a company. The contributions presented in this chapter show practically how an organisation can make specific improvements in this area, i.e., learn the design decision's impacts on the typically designed software and utilise the knowledge in tools supporting design, which is the main research question RQ1. The limitation of the presented improvement in this area is the fact that only very few design choices have been investigated. A larger set of investigated design alternatives would allow to collect more data and perform better comparison. Additionally, the tool support for quality-driven has not been used in a commercial context, which rendered it only a proof of a potentially useful concept.

Moreover, the selected improvements presented in this area show that a software organisation can internally identify design-related concerns, address them by empirical examination, and use the results for improvements in this area. By repeating this scenario an organisation should be able to build up knowledge that it needs for developing a software solution of higher quality, which partially answers the additional research question RQ2.

Furthermore, software design in a software organisation can be viewed in the context of mainstream quality improvement frameworks. For instance, CMMI for Development specifies the Technical Solution (TS) process area [124, p. 456]. According to CMMI for Development the purpose of Technical Solution is to '*design, develop, and implement solutions to requirements*' [124, p. 456]. Moreover, the Technical Solution area is to focus on [124, p. 456]:

- '*Evaluating and selecting solutions ... that potentially satisfy an appropriate set of allocated requirements*', in this context the selected design solutions provide alternatives to the final design solution and the tool support eases the selection process,
- '*Developing detailed designs for the selected solutions ...* ', also in this case the selected design solutions are concrete design examples that can be directly implemented, and naturally the selected design solutions do not constitute a whole system, but only a part of it,
- '*Implementing the designs as a product or product component*', the selected design solutions propose solutions only to specific problems that can be encountered in a system, hence they can only be a part of the implementation.

Therefore, based on the intended scope of the Technical Solution process area the selected design solutions as well as tools supporting design can be seen as being within the scope of this process area. Naturally, they do not fulfill all requirements of the TS area, but they support a subset of goals of this area within a major quality improvement framework. As a result, the proposed contributions in the area of development and selected design solutions are partially supporting a major quality improvement framework, i.e., CMMI for Development, which in this area answers additional research question RQ3.

# CHAPTER 7

# Related Work

In Chapters 2-6 we have discussed quality improvements in various areas of software organisation activities. In this chapter we present selected literature related to each improvement area discussed. Moreover we characterise how the existing research relates to the work presented in this thesis. The order of the literature review follows the order of the main contribution chapters, namely from Chapter 2 to 6.

## 7.1 Outsourcing Considerations

The outsourcing supplier selection and cooperation process have been presented in Chapter 2. Outsourcing in software development has been widely discussed in literature. General recommendations for outsourcing practices have already been defined. For instance, Donald J. Reifer presented seven outsourcing practices [104]. The recommendations focus on a business feasibility of possible outsourcing, establishing a good relationship with the outsourcing partner, measuring performance as well as using the results for stimulating the relation between the parties, and using outsourcing as an opportunity of knowledge transfer in both directions to and from the supplier. Finally, and probably most importantly, Reifer recommends to establish a win-win relationship between the organisations in order to sustain a healthy relationship. A creation of a good relationship with a outsourcing supplier is also a goal of the process described in Section 2.2.2. The building element of the relationship is the suitability of the supplier to the defined needs. Also the monitoring of the cooperation, described in Section 2.2, allows for taking corrective actions and developing the relationship with a supplier.

Additionally, high-level recommendations on developing outsourcing programmes have been presented by Power *et al.* [102]. Power *et al.* emphasise the importance of knowledge management in the process of building and improving an outsourcing programme. Furthermore, they specify an outsourcing management maturity model (OMMM) [102, p. 37]. These various high-level recommendations do not provide an actual recipe for how to outsource. However, they emphasise important aspects of developing and improving outsourcing programmes. As has been presented in Section 2.2 some of these recommendations can be found in the defined process for subcontractor selection and evaluation, e.g., knowledge buildup and learning process based on project experiences as well as aiming at partnership with the subcontractor.

A concrete set of criteria for supplier selection has been proposed by Christof

Ebert [33]. Ebert lists rules for supplier selection [33, p. 179-180]. These rules refer to the suitable size and business model of the supplier, the competence and internal processes of the supplier, and practical advice to obtain concrete names of people, who will be working from the supplier's side. Furthermore, Ebert presents a four-stage supplier agreement management. These recommendations and the agreement management plan are in line with the process presented in Section 2.2, but the process' details are adjusted to the specifics of the software service provider and the process is defined at a ready-to-use level.

Naturally, frameworks and methods have been proposed for aiding organisations in outsourcing planning and execution. For example, Hefley and Loesche proposed the eSourcing Capability Model for Client Organizations (eSCM-CL) [58, 59]. The eSCM-CL is an extensive collection of 95 best practices for client organisations of sourced services. The eSCM-CL specifies five life-cycle phases in sourcing [58, p. 29-32], namely:

- *Ongoing*, which focuses on management thought the whole sourcing life-cycle,
- *Analysis*, which focuses on identifying activities that could potentially be sourced,
- *Initiation*, which focuses on service provider selection and defining and formalising the sourcing relationship,
- *Delivery*, which focuses on actual service delivery by the service provider, and
- *Completion*, which focuses on ending a relationship with a sourcing service provider.

Furthermore there exist major quality improvement frameworks that specify practices for supplier selection and cooperation. For instance, CMMI for Acquisition Primer [105] defines selected practices for cooperation with suppliers and contractors in projects. Those practices are organised around various process areas, namely *Project Management*, Acquisition, and *Support*.

The outsourcing supplier selection and cooperation process presented in Chapter 2 is not based directly on any of the above-mentioned frameworks. However, it uses a certain subset of elements defined in them. For example, comparing to the eSCM-CL [58, 59] the subcontracting cooperation process, and its selection and evaluation phases, can be seen as a small subset of life-cycle phases defined in eSCM-CL, particularly, *Ongoing*, *Initiation*, and *Delivery*. However, the internal organisational considerations of decision making whether and what to outsource specified in the *Analysis* phase are not covered by the proposed process. Likewise, details of activities related to termination of an outsourcing relationship, as defined in the *Completion* phase, are not covered by the proposed process. Moreover, the selection and cooperation process described in this thesis focuses on a long term relationship. Therefore, the selection process is conducted infrequently, while cooperation happens in each case a project is implemented with the participation of a subcontracting partner.

## 7.2 Experiences with Agile Practices

Agile practices and practical recommendations on their usage have been presented in Chapter 3. In this chapter we review related reports on agile practices, and particularly Scrum use.

An important aspect of agile methods is communication. The impact of agile methods on internal and external communication in agile projects were investigated by Pikkarainen *et al.* [99]. Based on case studies, Pikkarainen *et al.* reported a generally positive impact of agile methods, including Scrum, on communication. They also report some negative impacts, e.g., an open space office that can positively influence informal face-to-face communication but can also be destructive in cases of performing tasks that require a high level of concentration [99, p. 328-329]. Similarly a positive impact of communication, particularly customer involvement, has been observed in experiences presented in Chapter 3. Additionally, as the case projects were often distributed ones, practices, e.g., common chat rooms in IM communicators, have been listed that aim at compensating the lack of physical proximity.

There are also reports on communication in distributed agile projects. For example, Korkala and Abrahamsson [75] report on case studies analysis where project practices were compared with recommended practices. Furthermore, reports on Scrum used in distributed projects exist. For example, Sutherland *et al.* [121] report on best practices for distributed teams as well as provide data on high productivity achieved in those teams. Complementary recommendations on practices in distributed Scrum teams were also presented by Paasivaara *et al.* [98]. These best practices and recommendations in many points overlap with the findings presented in Chapter 3. For instance, the use of teleconferencing or issue tracking tools, as well as the importance of building trust in the distributed team, have also been among best practice recommendations for distributed Scrum teams [98].

Moreover, tools aiming at improving communication and collaboration in distributed projects have been proposed. Bruegge *et al.* [21] suggest Sysiphus as a collaboration platform for system models. Sysiphus allows for collaboration on specific artifacts including traceability of important stakeholders, or notifications of changes. Such specialised tools for system model tracing have not been proposed. However, in recommendations presented in Chapter 3, the use of Wiki pages for collaborative documentation is advised.

In addition to reports on Scrum usage in various case studies, Scrum methodology has been investigated on compliance with mainstream SPI frameworks. Marçal *et al.* [83] have compared Scrum with CMMI and concluded that Scrum is a good starting point for companies at entry levels of CMMI. However, Marçal *et al.* add that organisations that are aiming at reaching a higher level of CMMI must use alternative practices to complement Scrum [83, p. 28]. Moreover, general compliance of agile principles with CMMI have been investigated by Glazer *et al.*, who state that '*Using Agile principles when designing and selecting CMMI practices can create more acceptable and appropriate process definition activities*' [50, p. 30]. Therefore, an organisation that decides to use agile development practices does not automatically prevent itself from complying with CMMI. These findings are especially important in the context of companies implementing their own SPIs and deciding to make improvements in

the area of project organisation, as presented in Chapter 3.

The Scrum-based projects that provided experiences resulting in the recommendations presented in Chapter 3 did not use any new methodologies. They rather adjusted or selected some of the above-mentioned methods and practices. It should be also noted that for an open and flexible organisation like Solita, Scrum and selected agile practices were suitable also from the organisation culture point of view, which may not be the case for all organisations.

## 7.3 Multi-site Development

In Chapter 4 we presented a specific practice used in multi-site development. However, other practices and general characteristics of multi-site development have already been discussed in many publications.

Wongthongtham *et al.* [135] discussed different advantages and challenges in multi-site development. They list, among others, the need for frameworks supporting communication and knowledge sharing [135, p. 355]. Additionally, they mention a need for technology tools that would assist in multi-site development [135, p. 357]. Both of these needs are addressed by the process and tool proposed and presented in Section 4.2.

The need for communication and more generally recognition of social and technical interdependencies in development teams were discussed by Amrit and van Hillegersberg [6]. They reviewed known problems of both software architecture and processes being altered during the life-cycle of a development project despite the use of best development practices. They recognise Socio-Technical patterns that link social dependencies with technical ones in a development process. They additionally propose a tool that can identify the dependency patterns based on communications between team members. Amrit's and van Hillegersberg's [6] work discusses possible problems in multi-site projects as well as possible solutions to the problems.

A practical solution to specific problems in multi-site software development was proposed by Lars Taxén [123]. The solution is based on practices in the telecommunication industry, namely one case organisation Ericsson. Taxén proposes the Integration Centric Development (ICD) approach consisting of three sub-processes. The focus of ICD is on promoting common understanding of a system developed at the functional level as well as increment and integration planning levels [123, p. 769]. Even though ICD has been used only in one organisation, the results were collected based on around 140 projects [123, p. 772] and they show that the method is valid for multi-site development where communication and common understanding are crucial [123, p. 779]. The ICD method is an example of an industry response to problems encountered in multi-site software development.

A specific problem of system architecture knowledge sharing has been presented by Ovaska *et al.* [97]. They report on industry case findings where it was found that simple coordination of development activities was not enough and suggest concentration on interdependencies between development activities [97, p. 244]. Ovaska *et al.* additionally propose architecture as means of communication about the system, which is particularly important in a distributed development environment. Addi-

tionally, Clerc *et al.* [29] have specified a method for assessing organisation compliance with defined architectural rules at multiple sites. Also Clerc *et al.* point out that the main problem in multi-site development is not the architectural rules, but the gathering and distribution of knowledge about those rules in an organisation. Another approach to architecture knowledge sharing has been presented by Victor Clerc [28], who collected architectural knowledge management practices as a set of patterns. Those patterns, similarly as in other publications, put strong emphasis on communication. The communication is promoted by practices and tools.

Finally, there are a number of industry case studies that describe experiences from multi-site software development initiatives. For example, Herbsleb *et al.* [60] report on experiences from nine projects carried out in Siemens Corporation. The report points out practical challenges and lessons learned in project coordination, development environment, and general communication, among others. An additional report on multi-site development practices at Siemens was presented by Bass *et al.* [15].

As can be seen from the number of various reports in literature, multi-site development adds up to the complexity of software development. There are, however, practices that aim at mitigating the effect of distribution in development projects. Moreover, concerns related to the architecture of the system developed were reported frequently as well as various methods for ensuring the architecture decisions communication and later enforcement.

The reoccurring themes of issues in multi-side development could be listed as follows:

- communication and knowledge sharing [135, p. 355],
- software architecture alterations caused by social and technical interdependencies [6], and
- development activities' interdependencies [97].

The architecture conventions ensuring process and tool supporting the rules validation (ARA), proposed in Chapter 4, addresses some of these issues. There exist also other tools that allow for validating software against certain architectural rules. For example, a commercial tool Lattix [1] can be used as a validation tool for architecture rules. In fact, Lattix provides additional architecture refactoring features that are not in the scope of the described process. Furthermore, there is an open source tool Macker [2] that has capabilities for checking rules defining package or class level references. Another tool that validates specific architectural rules was proposed by Selonen and Xu [113]. Their tool, called artDECO, uses UML profiles for architecture validation, which can be done at a lower level of granularity than packages, which are the granularity level utilised by ARA, e.g., at a class level.

Communication and knowledge sharing are encouraged in the process by formalising the communication between the different sites of the development team that implements and designs and validates the software (i.e., the required architecture decomposition rules, component mappings to the Java packages, and the binary

[1] `http://www.lattix.com`
[2] `http://innig.net/macker/index.html`

code). Additionally, the whole process aims at ensuring that the architecture is not altered by the multi-site character of the development, which is similar to concerns of social interdependencies. However, the architecture rules ensuring process does not adders any Socio-Technical patterns [6] as such. Finally, the development activities' interdependencies are addressed by the architecture rules ensurance process to a certain extent. Only the main development activities are organised in the process: the high level design, component design and implementation, and verification. The process does not address fine-grained activities.

## 7.4 Component Reuse

A open source component framework has been presented in Chapter 5. That framework aimed at selecting reusable components for commercial software solutions. However, component reuse is a well known concept aiming at optimising software development. The components can be in-house made, COTS components, or OSS components. The last category is of primary interest in this chapter. However, in order to put the OSS component selection in a broader context it will be useful to start from a brief overview of component reuse in commercial organisations.

Land *et al.* [77] have provided an extensive review of commercial off-the-shelf software (COTS) selection methods. The review covered 17 component selection methods from 1995 to 2006. In addition to the component selection methods Land *et al.* have conducted surveys, which included industry representatives, on the practically used selection methods. The results of that review provide a concrete industry point of view on the component selection. Lang *et al.* identified four processes in the selection methods, namely preparation, evaluation, selection, and supporting processes [77, p. 102]. The details of the processes were analysed and practical recommendations proposed, so that a custom selection process could be created. Finally, they provided four key recommendations that can be summarised as follows [77, p. 110]:

- different evaluation criteria should be used (i.e. functional, non-functional, architecture and business related criteria),
- evaluation should be an iterative process,
- architectural context of the whole system should be considered for compatibility and cost evaluation, and
- criticality of the components should be taken into consideration in the evaluation.

These findings based on COTS selection method review provide a good background for discussion of OSS component usage in commercial solutions. Moreover, the recommendations on usage of various criteria are fulfilled by the proposed OSS evaluation framework presented in Section 5.2. The framework utilises various criteria and sources of information, including open source community, available literature,

and the organisation's internal knowledge base in order to make a thorough evaluation. Naturally, evaluation of COTS and OSS components is not the same because the components differ significantly (e.g., in terms of offered support).

Additionally, Lang *et al.* have conducted a survey on component reuse in industry [78]. They have investigated development practices with reusable components, without the components, and development of such components. Their findings show some differences in the different development cases. For example, they have found that requirement specification is more flexible and accepts more changes in the case of development without reusable components compared to development with such components. Furthermore, the survey findings indicated acceptance of code changes even at the integration stage when a system was developed based on reusable components [78, p. 155]. This fact indicates a higher flexibility of solutions based on ready-made components, including OSS components that are of interest in this chapter.

The usage of OSS components poses concerns specific to the nature of OSS. For instance, Ruffin and Ebert [108] discussed legal concerns relating to Intellectual Property Rights (IPR) and license issues and they provided a few recommendations on how to use OSS components in commercial solutions safely. Moreover, they listed a few risks and benefits of OSS usage. For example, as benefits they mention short update and correction time for mainstream OSS projects, or in the case of mature OSS communities it can be less likely that the project is discontinued than in the case of small commercial component vendors. Moreover, they indicate that security of OSS components can be greater than in the case of commercial components where the code is not available for review by the whole community. At the same time, they mention a few risks attached to OSS components, for instance, a risk of breaching the license terms if the OSS component is not used carefully, or possible legal risks related to third party IPR violated in the OSS component. One of the suggested recommendations for reducing the legal risk was usage of packaging companies that shield the software development organisation from possible legal actions in the case of an OSS component breaching IPR. The evaluation framework presented in Section 5.2 also takes into consideration legal and licensing issues. Incompatible license for commercial use is one of the criteria in the *No Excuses* category that has to be fulfilled in order for the component to be even considered for use.

In more recent publications [32, 34] Christof Ebert revisited the subject of OSS in industry. The positives and negatives of using the OSS in a commercial context [32] are in line with those presented earlier [108]. An important addition made by Christof Ebert [32] is the analysis of different aspects of OSS that innovate software development in many ways, e.g., in terms of processes, technology, quality, architecture, standards used, and business model and marketing practices [32, p. 105-106]. However, organisations considering usage of OSS are still urged to consider their processes to be aligned with the nature of the OSS environment, e.g., frequent updates [32, p. 108]. Also other researches pointed out the importance of the context in which OSS is used. Ven *et al.* [129] presented results of a literature review and a case study based on 10 Belgian organisations. They have analysed five contradictory claims about OSS showing how differently specific aspects can be perceived. For instance, for organisations who only use the software, the access to the source

code may not be a clear benefit, especially if they do not have the competences to use it. Even though their focus was on infrastructure software, e.g., operating system, their findings showed the importance of evaluating OSS in the specific context of the business environment in which the OSS is used.

Ven and Verelst [128] studied a number of Belgian organisations in order to find out to what extent the organisations use external advice when adopting OSS. Also in this case the focus was on a general usage of OSS, rather than on a specific component reuse. However, the findings of the research showed that most of organisations use some form of commercial support. In this context methods for evaluating OSS in general are additionally important, especially if the company providing consulting services has the right tools for advising their clients.

The recommendations on considering the specific context for component usage [32, 129] are fulfilled by the OSS component evaluation framework (see Section 5.2) as the framework focus and used criteria are adjusted to a particular context in which the framework is intended to be used. On the other hand, the framework does not rely on any external help, as reported by Ven and Verelst [128], other than the available sources of information.

The effects of the availability of OSS components on software development have been discussed by Spinellis and Szyperski [119]. They note two important benefits generally associated with OSS components, i.e., the access to the source code and possibility to derive the developed code from the OSS. Additionally, they point out benefits of using OSS components, particularly the typically high quality of OSS components and the fact that they are potentially functionality richer than if developed in-house. However, they mention possible problems with the varying quality of different OSS components. For that problem, Spinellis and Szyperski suggest taking advantage of access to source code, mailing list archives, and bug tracking databases as indicators of OSS component quality and provided support [119, p. 30]. The concern of OSS component quality and ways of evaluating the quality based on OSS properties (e.g., the source code access) is particularly valid in the context of the evaluation framework discussed in Section 5.2. The quality of OSS components is the main focus of the framework as the evaluated components are to be used in commercial solutions. Moreover, sources of information that Spinellis and Szyperski [119] recommend, are included in the sources of information in the OSS evaluation framework.

Another interesting aspect of using OSS components was presented by Obrenović and Gašević, who discussed a problem of OSS component integration [133]. They addressed the integration problem of very heterogeneous OSS components by a middle-ware platform called Amico. The platform helped in quick integration of different OSS components using many standard interfaces and adapters, and a publish-subscribe way of communication [133]. This problem of component integration is also an aspect that must be taken into consideration when selecting components to be integrated with a commercial software solution.

The economics of OSS component usage in commercial projects have also been studied. Ajila and Wu [4] have reviewed literature and conducted interviews with companies' representatives in order to verify a few hypotheses about OSS component reuse. They concluded that a positive correlation between the OSS reuse and

software development economics existed. Additionally, they point out that no differences were noticed in the quality of OSS and proprietary software. Moreover, they indicate a strong positive impact of OSS component reuse on software quality, moderate impact on productivity and some impact on budget, in particular testing, platform, and documentation [4, p. 1526]. Therefore, economic reasons are valid reasons for using OSS components in commercial software solutions.

There has also been research on OSS utilisation based on selected quality factors, presented by Sohn and Mok [118]. They investigated a number of quality factors as specified in ISO/IEC 9126 that had the biggest influence on OSS component adoption by various user groups. Their findings showed that code sharing, functionality, and efficiency had the highest influence on the utilisation. This study showed what OSS-related factors may influence a user's decision on adoption of OSS components.

As can be seen based on this brief literature review, component reuse and particularly OSS usage in commercial solutions has been researched extensively. It can also be seen that the use or reuse of OSS should be considered in specific contexts. Additionally, the literature reports point out possible concerns related to OSS quality and ways how those concerns can be addressed. OSS components compared to COTS components offer access to the source code and additional community access that may be utilised in order to assess the quality of the component. The opportunities (i.e., community and source code access) that OSS components provide have been used in the evaluation framework presented in Section 5.2.

## 7.5 Software Design Considerations

The impact of selected design choices on performance in distributed applications as well as tool support for quality driven design have been presented in Chapter 6. As background to this broad topic of software design we will go through a design patterns and quality attributes overview in Section 7.5.1, and then briefly discuss tool support for incorporating and validating software design in Section 7.5.2.

### 7.5.1 Design Patterns and Quality Attributes Overview

Some of the quality attributes have been studied in depth in different technology contexts. For example, Cecchet *et al.* [26] have investigated performance of J2EE applications dependent on design and utilisation of the capabilities offered by the technology used. Similar research investigation for EJB has been done by Janne Mattila [86]. Other technology-specific performance investigations include, for example, EJB-specific performance issues presented by Gorton and Liu [52], Java RMI and .NET remoting [94] comparison by Elbers *et al.* [35], or object serialisation analysis by Hericko *et al.* [61]. Finally, there were reports on approaches that focused on predicting application performance already at the design level, for example as Liu *et al.* [80, 81] or Gomaa and Menascé [51] presented.

In this context the design considerations presented in Chapter 6 have been limited only to one quality attribute, namely performance. Moreover, instead of investigating the impact of the technology used, or design of the whole application, the

investigation was limited only to the design pattern used for a remote interface in a distributed application.

All these aspects of software design that designers should understand make them especially complex for everyday use in a software company. Naturally, design difficulty can be overcome by an experienced designer. However, as design is one of the activities constantly in use in a software company, contribution to the knowledge supporting design decisions is beneficial from the software quality point of view. In addition to explicit knowledge of a design solution, a software designer can utilise tools in design work. Therefore, let us next briefly discuss the role of various tools used in the software design process.

### 7.5.2 Tool Support for Incorporating and Validating Design Decisions

Tools supporting architecture design and decision tracking have already been proposed. For example, Babar and Gorton [8] have presented a framework and tool (PAKME) that stores architecture knowledge. The tool can serve as a repository of design decisions that can be reused in an organisation. Later PAKME was complemented by the AAKET tool, as presented by Aman-ul-haq and Babar [127]. AAKET can automatically and semi-automatically extract architecture-related knowledge from email messages and documents and store it in a repository, for example, PAKME. Another tool allowing for storing, tracking, and visualising architectural decisions in a form of graphs was a prototype tool called Archium, as presented by Jansen *et al.* [64]. Furthermore, a different perspective on software system architecture has been presented by Garlan *et al.* [47], who focus on architecture evolution. Garlan *et al.* [47] specify an evolution style as a pattern of changes, which can be domain-specific, in system architecture over time. They present the Ævol tool, which supports analysis and planning for system architecture evolution.

In addition to tools supporting system architecture design, there are tools that focus on component or subsystem design. Florijn *et al.* [38] presented a tool prototype that supports a designer in using design patters. The tool supported pattern generation, integration of patterns with existing code, and pattern validation [38, p. 474]. The tool was a prototype for the Samlltalk environment. However, it showed what can be expected from such tools. A more recent example of a tool proposal for design support has been presented by Hammouda *et al.* [56]. The tool, which is called INARI, supports generation and validation of certain design pattern instances based on rules expressed as tool-specific patterns.

Finally, there are tools and models aiming at supporting the design process with respect to quality attributes. For example, at the analysis stage a model that aims at estimating quality attributes was presented by Janakiram and Rajasree [63]. Their quality estimation model uses an analysis model for estimating specific quality attributes. Moreover, methods for verification or analysis of existing software implementations have also been proposed. One example could be the 'Metric-Driven Analysis and Feedback' system (Amadeus) presented by Selby *et al.* [112]. The Amadeus system provides feedback about the system status based on various metrics, development process stage, and even history data. All this information can be used to

identify potentially 'problematic' parts of a system. Another approach has been proposed by Radu Marinescu [85], who suggested 'detection strategies' that abstract code metrics into a user-friendly form in order to detect problems and their causes in code.

The tool support for quality driven design (i.e., MADE) in Chapter 6 was used only as an example tool and its functionality was limited to suggesting three design choices to a designer based on the design pattern impact on performance of distributed applications. The use of an existing tool for such a purpose showed another way of using this particular tool, which later was extended to a tool called INARI [56]. For the design selection the tool used the information obtained from external sources and required manual configuration. Finally, this tool support was meant for new systems under development, unlike the problem detection strategies for existing code [85].

All these tool examples show the richness of tools used for design decision implementation and validation. Software development organisations can potentially use tools for different purposes during development.

# CHAPTER 8

# Conclusions

We have discussed five areas of quality improvements proposed in this thesis (see Figure 8.1). These improvement areas are summarised in Section 8.1. In this summary the research questions formulated in Chapter 1 (see page 6) and addressed for individual quality improvement areas in Chapters 2-6 are revisited. The research questions are considered in the light of all contributions presented in previous chapters. Additionally, limitations of this thesis are highlighted. Next, this thesis author's contributions to the included publications are presented in Section 8.2. Finally, possible directions of future research in the context of this thesis are presented in Section 8.3.

## 8.1 Summary

In this thesis we have presented five different areas of quality improvements in a software organisation. The areas were categorised into two categories of *Organisation* and *Development*.

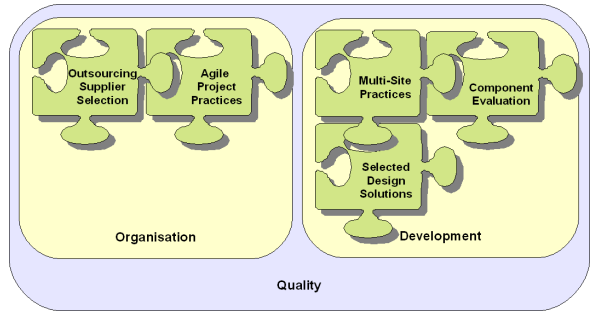



**Figure 8.1** Selected quality improvement areas

The improvement areas in the *Organisation* category focused on activities in a software organisation that do not directly result in producing software, but which they impact the software production indirectly. This category included improvements in *Outsourcing Supplier Selection* and *Agile Project Practices*. For the *Outsourcing Supplier Selection* area a process for subcontractor selection and evaluation was proposed, which constituted a cooperation process with subcontractors. The selection of subcontracting partners was based on a number of criteria and multiple elimination steps. The selection process has been used in the case company Solita for selecting subcontracting partners. Moreover, for evaluation of subcontractor work a set of metrics and criteria were proposed that can be used relatively easily in commercial settings. The criteria in their extended version were further used for comparing Scrum and non-Scrum projects, which was presented in relation to *Agile Project Practices*. In the

area of specific *Agile Project Practices* it was showed based on data obtained from 18 Scrum and non-Scrum projects that Scrum projects generally performed better than other project types. Therefore, Scrum methodology was recommended as a preferred project methodology. Additionally, based on reported experiences a number of practices and tools was listed that were found useful in work with subcontractors.

The quality improvements grouped in the *Development* category focused on activities directly related to software development and included: *Multi-Site Practices*, *Component Evaluation*, and *Selected Design Solutions*. In all of these areas specific quality improvements were proposed that improved the quality of software solutions developed.

In the area of *Multi-Site Practices* a specific process and tool were proposed, which constitute a practice supporting development in a multi-site environment. The practice focused on ensuring architectural rules in software solutions development in multiple development sites by different organisations, i.e., outsourcing partners.

In addition to the multi-site development practices, an OSS component evaluation framework was proposed, as an improvement in the *Component Evaluation* area. The framework has been constructed to select OSS components that can be reused in software solution lines taking into account possible risks related to using a component. The framework was using multiple sources of information in order to get a full picture of the evaluated OSS component. The framework became part of the evaluation process used at the case company Solita.

Finally, three design patterns and their influence on performance of distributed systems were investigated, as an improvement in the *Selected Design Solutions* area. The results provided recommendations on optimal usage of the design patterns for implementation of an application's remote interfaces. The recommendations take into account complexity of the interface, number of calls, and the technology used for data serialisation. Moreover, based on the recommendations a tool support for quality-driven design was proposed. The tool guided the designer to choose the most suitable design solution based on selected quality requirements, performance and flexibility, in this case for the interface.

Having reviewed all the quality improvements in different areas of a software organisation's operations, it is possible to look at the research questions in the context of the whole thesis.

### 8.1.1 Research Questions Revisited

In this thesis three research questions were formulated. The questions were addressed partially already in individual chapters discussing the five quality improvement areas. Now the questions can be revisited in the context of the findings gathered throughout the thesis.

**RQ1: How can small quality improvements in different areas of a software company's operation help to improve software solution quality?**

The light-weight improvements affect directly or indirectly the quality of software produced in a software organisation. The various effects within the five different

areas of improvements have been presented. The effects of improvements were demonstrated in each area.

The careful *Outsourcing Supplier Selection* process ensured that the organisations and people developing software for end customers were suitable for that task. The usage of suitable partners for the specific organisational setting ensured that the end software solution and consequently the customer was not affected by changes in project organisation and staffing. The *Agile Project Practices* provided data supporting usage of Scrum methodology as the preferred methodology that improved the success of projects. Additionally, the *Agile Project Practices* provided concrete recommendations for tools and practices that made work with subcontractors efficient. The practices and tools were transparent for end customers and ensured that regardless of project organisations the practices selected were appropriate for developing a software solution.

The usage of *Multi-Site Practices* that are adjusted to the project environment improves the quality of final software solutions produced. In the presented example the architectural decisions were verified in a multi-site environment. Therefore, for the end customer the quality of the software solutions is unaffected because of usage of a multi-side project arrangement with subcontracting partners. Moreover, *Component Evaluation* improves the quality of software solutions where ready-made OSS components are used. The evaluation framework allows eliminating OSS components that have inadequate quality for usage in commercial solutions. Finally, the design improvements within the *Selected Design Solutions* area demonstrated how selected quality attributes can be improved and linked with specific design decisions that are made frequently when software solutions are developed.

In the context of a software company, an initiative resulting in improvements in the quality of a software solution means that the quality of the main product of the organisation has been affected. The proposed quality improvements in the *Development* category had a very local scope, e.g., design improvements. However, as they focused on improving a part of the software developed, they also had to affect the overall solutions. The improvements categorised in the *Organisation* category did not have a strong and direct link to overall software solution quality. However, they all focused on activities directly supporting software development, e.g., the competence level of subcontracting partners who were implementing the software. Hence, such initiatives also had an impact on the software produces. Therefore, it can be argued that the proposed improvements can affect the overall software solutions developed in the software organisation. However, the exact impact of one improvement over another has not been quantitatively compared.

Naturally, the proposed quality improvements have a very limited scope of the improvements. However, within their scope they improve quality. Furthermore, the improvement areas propose small and light-weight improvements that are relatively easy to implement for a software organisation. As many different improvement areas were proposed based on real needs observed in a case organisation, other organisations should be able to select at least a subset of areas that are applicable in their environment.

**RQ2: How can a small or medium-sized software service company implement a flexible and light-weight quality improvement initiative based on its internal organisational experience?**

As was presented a software organisation is able to react to changes in their environment and seek solutions that solve new problems. An organisation can actively prepare for upcoming changes, as in the case of the *Outsourcing Supplier Selection* area, and be prepared for changes. The presented five areas of quality improvement are only examples of possible improvement areas and they definitively do not fulfill all possible improvements in any company. Moreover, as it was clarified in Section 1.4, the presented improvements were not introduced as an organised improvement programme, but they were introduced as a grass-root level improvement initiative.

The individual improvements provided quality improvements in the specific areas. It was also presented how a company can develop measures improving a given area, i.e., improvements by changes in processes, tools, organisations, etc. The list of improvement areas presented in this thesis is definitely incomplete. However, a complete list of any possible improvements was not a goal of this thesis. This thesis presents possibilities and a path for quality improvements driven by internal needs of the software organisation. Based on the five areas, an other organisation should be able to notice areas where they can introduce improvements. They can be areas overlapping with those presented in this thesis. However, there are likely to be new area or similar areas but with a different context.

Additionally, a quality improvement initiative that originates at a grass-root level of the organisation requires the organisation to encourage such initiatives. Individuals, or groups of individuals, working on an everyday basis with software development can be encouraged to make suggestions for observed quality improvement needs. The encouragement can be given, for example, as a possibility to 'make a difference' in the organisation and see that at least some of the suggestions are implemented, which was the case of improvements presented in this thesis.

Therefore, this thesis showed an example of a possible in-house quality improvement initiative focusing on various areas of software organisation operations. The concrete examples of improvements and ways how they can be implemented provide practical guidance within the scope of the selected areas. The examples can be used as a starting point and inspiration for further improvements in other organisations. Therefore, similar medium-sized software service providers can internally select areas that they find important in their organisations. For example, in the case of design they can choose different design solutions to be investigated, which could be suggested by designers or architects. Some areas can be irrelevant for them, for example, the outsourcing partner selection if they do not use outsourcing partners. However, such organisations can use the experiences presented in this thesis and adapt and extend them for their organisations' needs.

It should be also noted that the grass-root approach to quality in software organisations may be limited by the organisational culture and readiness of such organisations to encourage organisation members to take the initiative. Additionally, even though it is likely that similar organisations as the case organisation can be inspired

to start the grass-root quality improvement approach, this approach has not been validated in other organisations.

**RQ3: How can the flexible and light-weight approach to quality improvement complement an official SPI framework?**

This question aimed at highlighting consequences for a software organisation of using a flexible and in-house experience-based quality improvement initiative for their possible future choices. If a software organisation finds it suitable to implement their own quality improvement initiative, the organisation may in the future still decide to implement an official (i.e., certified) SPI programme. This question was addressed partially for each individual improvement area by finding links to a major quality improvement framework. As an example of a framework CMMI for Development [124] was chosen. The links between the presented five quality improvement areas and selected CMMI for Development process areas are summarised in Table 8.1.

**Table 8.1** Links between the five quality improvement areas and CMMI for Development

| Contribution area | CMMI for Dev. area | Chapter |
|---|---|---|
| *Organisation* | | |
| Outsourcing | | |
| Supplier Selection | Supplier Agreement Management (SAM) | Chapter 2 |
| Agile Project Practices | Integrated Project Management (IPM) | Chapter 3 |
| *Development* | | |
| Multi-Site Practices | Verification (VER) | Chapter 4 |
| Component Evaluation | Technical Solution (TS) and Risk Management (RSKM) | Chapter 5 |
| Selected Design Solutions | Technical Solution (TS) | Chapter 6 |

Table 8.1 gathers the CMMI for Development process areas that were linked with the given quality improvement area in Chapters 2-6. The links are based on the main goals of the process areas and particular focus of the improvement areas. The links present only partial fulfillment of a specific CMMI for Development process area by a particular quality improvement area. The links have been established, but by no means do the proposed improvements in five areas fulfill the process areas of CMMI for Development. They fulfill only a subset of goals of a given process area. To be more precise, in this thesis we have identified relations of the five proposed improvement areas to six CMMI for Development areas out of 22. That means that an organisation addressing the proposed five improvement areas would partially fulfill only about one quarter of all CMMI for Development areas.

The links between the CMMI for Development and the five improvement areas show that even unofficial improvement initiatives may be complementary to a major

quality improvement framework. Therefore, even though the proposed five areas for quality improvement do not fulfill all requirements of CMMI for Development they still support it partially. That implies that a choice of a in-house quality improvement initiative does not automatically mean that the software organisation cannot apply in the future an official quality improvement programme. Naturally, in order to fulfill all requirements of different levels of compliance with CMMI for Development, the quality improvements in various areas would have to be accompanied with additional improvements fulfilling CMMI requirements.

### 8.1.2 Limitations

There are a few limitations of the work presented in this thesis. First, the presented quality improvements were implemented only in one software organisation. The fact that only one organisation was involved poses limits to possible generalisation of the findings. In order to make any wider recommendations, i.e., create a framework or formal quality improvement programme for medium-size software organisations, experiences from a larger number of software organisations would be needed. Therefore, as has been already mentioned, the proposed quality improvements in the five areas of company operations should not be treated as a universal and defined quality improvement programme.

Moreover, the five individual quality improvement areas have their own limitations. Partially they derive from the fact that only one organisation is discussed in this thesis and the improvements were not applied to other organisations. However, there are also additional limitations. For example, in the case of selected design solutions, the proposed improvements are limited only to three design patterns and two technologies, while in software organisations the variety of design decisions is much wider. Therefore, all those limitations should be taken into consideration when applying the quality improvements in other organisations.

Additionally, the extent of the available data did not in all cases allow for proper statistical analysis. This fact implies possible error included in the data. Naturally, in all the cases the data was collected with as much care as possible. Furthermore, not all data was possible to be measured, and had to be interpreted. For example, in the case of agile practices recommendations based on the conducted interviews.

Finally, an in-house grass-root level quality improvement initiative does not give guarantee that the selected areas for improvement will be implemented in the organisation, or that they will be widely used. For example, the tool support for quality driven design suggested in this thesis was a successful proof of concept, but in the particular context of the organisation it was not feasible. Therefore, the suggestions for quality improvements should be considered in a wider context in the organisation.

## 8.2 Author's Contributions in Publications

The author of this thesis made contributions to the included publications as follows.

**Publication [P1]** describes the cooperation process with subcontracting partners and provides practical recommendations on tools and practices useful in working with subcontractors. This publication is based on the cooperation process presented in publication [P4] and extends findings presented in publication [P2] with details and practical tips.

The author formulated the practical recommendations based on data collected as well as arranged the findings in a way that presents the subcontractor cooperation process in a practitioner-oriented fashion.

**Publication [P2]** describes findings of differences between Scrum and non-Scrum projects based on 18 commercial projects. The Goal Question Metric approach was used as the research methodology. The results are based on qualitative and quantitative analysis of data collected. The proposed recommendations aim at helping projects in a similar commercial context to choose best agile practices.

The author designed and performed interviews with project managers. Additionally, the author collected data from IT systems and analysed the findings. The author formulated recommendations based on all the collected data.

**Publication [P3]** describes a framework for evaluation of OSS components for reuse in multiple commercial software solutions. The framework is based on a selected set of quality criteria that were found most relevant and feasible to be used in a commercial setting. The evaluation framework is also presented in the context of Software Product Lines creation. Finally, feedback from the framework usage is presented.

The author collected evaluation criteria and performed criteria selection together with Kimmo Kiviluoma and Tero Poikonen. The author together with Imed Hammouda conceptualised the evaluation process with a relation to SPL construction. The author presented the process as usable instructions used by evaluators. The author collected and analysed data obtained from evaluation performed and received feedback on the evaluation process.

**Publication [P4]** describes a process for selecting and evaluating subcontractors for cooperation in a Software Service Provider. Sets of particular criteria for both purposes are selected. The criteria include specific metrics and qualitative criteria. Finally, results from the process usage are reported.

The author specified the selection process as well as executed it together with Karri Mustonen. The author collected data from the selection process. The author specified the process and criteria for evaluation of subcontractors' work.

**Publication [P5]** describes a process and tool for ensuring architectural rules in multi-site development. It is argued that software development distributed between development sites should use a validation process that ensures that architectural rules are not altered during the development. The proposed validation process is supported by a specific tool called ARA. Evaluation of the process and tool was performed and results reported.

The author implemented the ARA tool used for architectural rules validation in Java. The author sketched the architectural rules assurance process, and finalised it together with the co-authors. The author executed test cases and analysed the results.

**Publication [P6]** extends the work presented in publication [P8] by performance results collected for the selected design patterns implemented in .NET technology. Three design patterns used as an interface for remote communication are investigated for their influence on overall application performance. The additional results from the .NET investigation are combined with the earlier results from the J2EE technology. General conclusions are drawn on the design aspects of remote communication that affect application performance.

The author ported the test application from J2EE implementation to .NET implementation. Additionally, the author executed tests measuring application performance as well as performed additional tests of different serialisation variants. Finally, the author analysed the results.

**Publication [P7]** describes a usage of the MADE tool [55] as a tool for quality-driven design. The tool was configured in a way that allows a designer to choose between design solutions based on a specific quality attribute. The supported quality attributes were performance and flexibility of interface used for remote communication in a software system. The solution choice recommendations were based on empirical data gathered in publication [P8].

The author used results of performance investigations from publication [P8] as data for possible design choices. The author modelled design choices in the MADE tool [55] together with Imed Hammouda.

**Publication [P8]** described results of investigation on how three design patterns used for implementing a remote interface affect performance of a J2EE application. The selected design patterns were implemented in a fully-functional test application. A number of tests were executed and performance-related metrics collected. The results showed performance differences between the design patterns used for remote communication.

The author implemented test application in J2EE, designed and executed a number of tests and analysed the data.

## 8.3 Future Work

The future work can be seen at different levels. At one level the work may concentrate on further improvements within the selected improvement areas. For example, the proposed tools can be further improved making them easier to use, or supporting new technologies. Furthermore, new improvement areas can be selected and improvements proposed. For example, improvements in software solution line development in a way that reflects even closer the SPL development.

Another level of future research can include investigation of other organisations for evaluating the proposed improvements in another organisation. Additionally, it could be investigated whether grass-root level quality improvement initiatives can be applied only in organisations with specific organisational cultures.

Generally, as a quality improvement is an ongoing process and software organisations are very dynamic structures, there should be many new possible research directions available.

# Bibliography


[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods. review and analysis. http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf, 2002. VTT Publications : 478.

[2] Inc. Advanced Development Methods. Controlled chaos : Living on the edge. `http://www.controlchaos.com/download/Living%20on%20the%20Edge.pdf`, 1996.

[3] Faheem Ahmed, Luiz Fernando Capretz, and Muhammad Ali Babar. A model of open source software-based product line development. *32nd Annual IEEE International Computer Software and Applications Conference*, 0:1215–1220, 2008.

[4] Samuel A. Ajila and Di Wu. Empirical study of the effects of open source adoption on software development economics. *J. Syst. Softw.*, 80(9):1517–1529, 2007.

[5] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall / Sun Microsystems Press, 2001.

[6] Chintan Amrit and Jos van Hillegersberg. Detecting coordination problems in collaborative software development environments. *Information Systems Management*, 25(1):57–70, 2008.

[7] Atos Origin. Qsos version 1.6. `http://www.qsos.org/?page_id=3`, 23 October 2006.

[8] Muhammad Ali Babar and Ian Gorton. A tool for managing software architecture knowledge. In *SHARK-ADI '07: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, page 11, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Felix Bachmann and Len Bass. Introduction to the attribute driven design method. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 745–746, Washington, DC, USA, 2001. IEEE Computer Society.

[10] Felix Bachmann, Len Bass, and Mark Klein. Moving from quality attribute requirements to architectural decisions. In *STRAW'03 : Second International*

*SofTware Requirements to Architectures Workshop located at ICSE'03*, pages 122–130, Portland, OR, USA, 2003.

[11] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, and Charles B. Weinstock. Quality attributes. Technical Report CMU/SEI-95-TR-021, Software Engineering Institute, Carnegie Mellon University, December 1995.

[12] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. `ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf`, 1994. accessed in March 2009.

[13] Len Bass, Mark Klein, and Felix Bachmann. Quality attribute design primitives. Technical Note CMU/SEI-2000-TN-017, Software Engineering Institute (SEI), 2000.

[14] Leonard J. Bass, Mark Klein, and Felix Bachmann. Quality attribute design primitives and the attribute driven design method. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 169–186, London, UK, 2002. Springer-Verlag.

[15] Matthew Bass, James D. Herbsleb, and Christian Lescher. Collaboration in global software projects at siemens: An experience report. In *ICGSE '07: Proceedings of the International Conference on Global Software Engineering*, pages 33–39, Washington, DC, USA, 2007. IEEE Computer Society.

[16] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick andRobert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. http://agilemanifesto.org/, 2001.

[17] Izak Benbasat, David K. Goldstein, and Melissa Mead. The case research strategy in studies of information systems. *MIS Quarterly*, 11(3):369–386, 1987.

[18] Barry Boehm. A view of 20th and 21st century software engineering. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 12–29, New York, NY, USA, 2006. ACM.

[19] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[20] Business Readiness Rating. http://www.openbrr.org/, 2008.

[21] Bernd Bruegge, Allen H. Dutoit, and Timo Wolf. Sysiphus: Enabling informal collaboration in global software development. In *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 139–148, Oct. 2006.

[22] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerland, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[23] Capgemini. Open Source Maturity Model. http://www.osspartner.com/portail/sections/accueil-public/evaluation-osmm, 2003.

[24] Erran Carmel and Pamela Abbott. Why 'nearshore' means that distance matters. *Commun. ACM*, 50(10):40–46, 2007.

[25] Aileen Cater-Steel, Wui-Gee Tan, and Mark Toleman. Challenge of adopting multiple process improvement frameworks. 2006.

[26] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of EJB applications. In *17th ACM Conference on ObjectOriented Programming*, pages 246–261, Seattle, Washington, 2002.

[27] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[28] Viktor Clerc. Towards architectural knowledge management practices for global software development. In *SHARK '08: Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, pages 23–28, New York, NY, USA, 2008. ACM.

[29] Viktor Clerc, Patricia Lago, and Hans van Vliet. Assessing a multi-site development organization for architectural compliance. In *WICSA '07: Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, page 10, Washington, DC, USA, 2007. IEEE Computer Society.

[30] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.

[31] Tore Dybå. Factors of software process improvement success in small and large organizations: an empirical study in the scandinavian context. *SIGSOFT Softw. Eng. Notes*, 28(5):148–157, 2003.

[32] Christof Ebert. Open source drives innovation. *IEEE Softw.*, 24(3):105–109, 2007.

[33] Christof Ebert. Optimizing supplier management in global software engineering. In *ICGSE '07: Proceedings of the International Conference on Global Software Engineering*, pages 177–185, Washington, DC, USA, 2007. IEEE Computer Society.

[34] Christof Ebert. Open source software in industry. *IEEE Software*, 25:52–53, 2008.

[35] Willem Elbers, Frank Koopmans, and Ken Madlener. Java RMI and .NET remoting performance comparison. At URL http://www.niii.ru.nl/~marko/onderwijs/oss, December 2004. Radboud Universiteit Nijmegen.

[36] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. *UML 2 Toolkit*. John Wiley and Sons, October 2003.

[37] Thomas Erl. *SOA Design Patterns*. Prentice Hall PTR, 2009.

[38] Gert Florijn, Marco Meijers, and Pieter van Winsen. Tool support for object-oriented patterns. In *ECOOP'97 Object-Oriented Programming*, volume 1241/1997 of *Lecture Notes in Computer Science*, pages 472–495. Springer Berlin / Heidelberg, 1997.

[39] International Organization for Standardization. ISO/IEC 15504, Information technology - Process assessment, part 1 to part 5, 1998-2005.

[40] International Organization for Standardization. Iso 9001:2000 quality management systems – requirements, 12 2000.

[41] International Organization for Standardization. Iso 9000:2005 quality management systems – fundamentals and vocabulary, 09 2005.

[42] International Organization for Standardization. Iso 9004:2009 managing for the sustained success of an organization – a quality management approach, 10 2009.

[43] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Menlo Park, 1997.

[44] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[45] Martin Fowler. Continuous integration. `http://martinfowler.com/articles/continuousIntegration.html`, May 2006.

[46] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.

[47] David Garlan, Jeffrey M. Barnes, Bradley Schmerl, and Orieta Celiku. Evolution styles: Foundations and tool support for software architecture evolution. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 European Conference on Software Architecture 2009*, Cambridge, UK, 14-17 September 2009.

[48] David A. Garvin. What does "product quality" really mean? *MIT Sloan Management Review*, 26(1):25–43, 1984.

[49] R. L. Glass, I. Vessey, and V. Ramesh. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44(8):491 – 506, 2002.

[50] Hillel Glazer, Jeff Dalton, David Anderson, Michael D. Konrad, and Sandra Shrum. Cmmi or agile: Why not embrace both! Technical Report CMU/SEI-2008-TN-003, Software Engineering Institute, Carnegie Mellon University, November 2008.

[51] Hassan Gomaa and Daniel A. Menascé. Design and performance modeling of component interconnection patterns for distributed software architectures. In *WOSP '00: Proceedings of the 2nd international workshop on Software and performance*, pages 117–126, New York, NY, USA, 2000. ACM.

[52] Ian Gorton and Anna Liu. Evaluating the performance of ejb components. *IEEE Internet Computing*, 7(3):18–23, 2003.

[53] Mark Grand. *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, volume 1. Wiley, 1998.

[54] Imed Hammouda, Juha Hautamäki, Mika Pussinen, and Kai Koskimies. Managing variability using heterogeneous feature variation patterns. In *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005*, pages 145–159, 2005.

[55] Imed Hammouda, Johannes Koskinen, Mika Pussinen, Mika Katara, and Tommi Mikkonen. Adaptable concern-based framework specialization in UML. In *ASE*, pages 78–87, 2004.

[56] Imed Hammouda, Anna Ruokonen, Mika Siikarla, André L. Santos, Kai Koskimies, and Tarja Systä. Design profiles: toward unified tool support for design patterns and uml profiles. *Softw. Pract. Exper.*, 39(4):331–354, 2009.

[57] Oyvind Hauge, Thomas Osterlie, Carl-Fredrik Sorensen, and Marinela Gerea. An empirical study on selection of open source software - preliminary results. In *FLOSS '09: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 42–47, Washington, DC, USA, 2009. IEEE Computer Society.

[58] Bill Hefley and Ethel Loesche. esourcing capability model for client organizations (escm-cl), escm-cl v1.1, part 1. `http://www.itsqc.org/downloads/documents/eSCM-CL_v1.1_Part1.pdf`, September 27 2006.

[59] Bill Hefley and Ethel Loesche. esourcing capability model for client organizations (escm-cl), escm-cl v1.1, part 2. `http://www.itsqc.org/downloads/documents/eSCM-CL_v1.1_Part2.pdf`, September 27 2006.

[60] James D. Herbsleb, Daniel J. Paulish, and Matthew Bass. Global software development at siemens: experience from nine projects. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 524–533, New York, NY, USA, 2005. ACM.

[61] Marjan Hericko, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec, and Ales Zivkovic. Object serialization analysis and comparison in java and .NET. *SIGPLAN Not.*, 38(8):44–54, 2003.

[62] IT Governance Institute (ITGI). Control objectives for information and related technology - cobitt 4.1. `http://www.isaca.org/Template.cfm?Section=COBIT6&Template=/TaggedPage/TaggedPageDisplay.cfm&TPLID=55&ContentID=7981`, January 2010.

[63] D. Janakiram and M. S. Rajasree. Request: Requirements-driven quality estimator. *SIGSOFT Software Engineering Notes*, 30(1):4, 2005.

[64] A. Jansen, J. van der Ven, P. Avgeriou, and D.K. Hammer. Tool support for architectural decisions. In *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, pages 4–4, Jan. 2007.

[65] Pertti Järvinen. *On Research Methods*. Tampereen yliopistopaino Oy, Tampere, Finland, 2004.

[66] Jr. Jay F. Nunamaker and Minder Chen. Systems development in information systems research. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, volume 3, pages 631 – 640. IEEE, Jan 1990.

[67] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[68] Karlheinz Kautz. Software process improvement in very small enterprises: does it pay off? *Software Process: Improvement and Practice*, 4(4):209226, 1998.

[69] Karlheinz Kautz, Henrik Westergaard Hansen, and Kim Thaysen. Applying and adjusting a software process improvement model in practice: the use of the ideal model in a small software enterprise. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 626–633, New York, NY, USA, 2000. ACM.

[70] Karlheinz Kautz, Henrik Westergaard, and Kim Thaysen. Understanding and changing software organisations: an exploration of four perspectives on software process improvement. *Scand. J. Inf. Syst.*, 13:31–50, 2001.

[71] Rick Kazman. Tool support for architecture analysis and design. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops*, pages 94–97, New York, NY, USA, 1996. ACM.

[72] Nazrina Khurshid, Paul L. Bannerman, and Mark Staples. Overcoming the first hurdle: Why organizations do not adopt cmmi. In *ICSP '09: Proceedings of the International Conference on Software Process*, pages 38–49, Berlin, Heidelberg, 2009. Springer-Verlag.

[73] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. *IEEE Softw.*, 13(1):12–21, 1996.

[74] Alan S. Koch. *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House Publishers, 2005.

[75] M. Korkala and P. Abrahamsson. Communication in distributed agile development: A case study. pages 203 –210, aug. 2007.

[76] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, 1995.

[77] Rikard Land, Laurens Blankers, Michel R. V. Chaudron, and Ivica Crnkovic. Cots selection best practices in literature and in industry. In *ICSR*, pages 100–111, 2008.

[78] Rikard Land, Daniel Sundmark, Frank Lüders, Iva Krasteva, and Adnan Causevic. Reuse with software components - a survey of industrial state of practice. In *ICSR*, pages 150–159, 2009.

[79] N. G. Lester, F. G. Wilkie, D. McFall, and M. P. Ware. Investigating the role of cmmi with expanding company size for small- to medium-sized enterprises. *Software Process: Improvement and Practice*, 2009. http://dx.doi.org/10.1002/spip.409.

[80] Yan Liu, Alan Fekete, and Ian Gorton. Predicting the performance of middleware-based applications at the design level. In *WOSP '04*, pages 166–170, New York, NY, USA, 2004. ACM Press.

[81] Yan Liu, Alan Fekete, and Ian Gorton. Design-level performance prediction of component-based applications. *IEEE Trans. Softw. Eng.*, 31(11):928–941, 2005.

[82] Michael Mahemoff. *Ajax Design Patterns*. O'Reilly Media, 2006.

[83] Ana Sofia C. Marçal, Bruno Celso C. de Freitas, Felipe S. Furtado Soares, Maria Elizabeth S. Furtado, Teresa M. Maciel, and Arnaldo D. Belchior. Blending scrum practices and cmmi project management process areas. *Innovations in Systems and Software Engineering*, 4(1):17–29, April 2008.

[84] Floyd Marinescu. *EJB Design Patterns*. The MiddleWare Company, 2002.

[85] R. Marinescu. Measurement and quality in object-oriented design. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 701–704, Sept. 2005.

[86] Janne Mattila. EJB Performance. Master's thesis, Tampere University of Technology, 2004.

[87] Deepti Mishra and Alok Mishra. Software process improvement methodologies for small and medium enterprises. In *PROFES*, pages 273–288, 2008.

[88] Deepti Mishra and Alok Mishra. Software process improvement in SMEs: A comparative view. *Computer Science and Information Systems*, 6(1):111–140, 2009.

[89] Taichi Muraki and Motoshi Saeki. Metrics for applying gof design patterns in refactoring processes. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 27–36, New York, NY, USA, 2001. ACM.

[90] Navica. Open Source Maturity Model. `http://www.navicasoft.com/pages/osmm.htm`, 2008.

[91] Eila Niemelä and Anne Immonen. Capturing quality requirements of product family architecture. *Information & Software Technology*, 49(11-12):1107–1120, 2007.

[92] Linda M. Northrop. Software product line adoption roadmap. Technical Report CMU/SEI-2004-TR-022, ADA431117, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2004.

[93] Linda M. Northrop, Paul C. Clements, Felix Bachmann, John Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, Lawrence Jones, Robert Krut, Reed Little, John McGregor, and Liam O'Brien. A framework for software product line practice, version 5.0. `http://www.sei.cmu.edu/productlines/framework.html`, July 2007. Software Engineering Institute, Carnegie Mellon University.

[94] Piet Obermeyer and Jonathan Hawkins. Microsoft .NET remoting: A technical overview. At URL `http://msdn.microsoft.com/library`, July 2001. Microsoft Corporation.

[95] Office of Government Commerce (OGC). It service management - itil. `http://www.best-management-practice.com/IT-Service-Management-ITIL/?trackid=002192`, January 2010.

[96] Pirkko Östring. *Profit-Focused Supplier Management: How to Identify Risks and Recognize Opportunities*. Amacom, 2003.

[97] Päivi Ovaska, Matti Rossi, and Pentti Marttiin. Architecture as a coordination tool in multi-site software development. *Software Process: Improvement and Practice*, 8(4):233–247, 2003.

[98] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Using scrum in a globally distributed project: a case study. *Software Process: Improvement and Practice*, 13(6):527–544, 2008.

[99] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008.

[100] Francisco J. Pino, Félix García, and Mario Piattini. Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Control*, 16(2):237–261, 2008.

[101] C. Potts and G. Bruns. Recording the reasons for design decisions. In *ICSE '88: Proceedings of the 10th international conference on Software engineering*, pages 418–427, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[102] Mark J. Power, Kevin C. Desouza, and Carlo Bonifazi. Developing superior outsourcing programs. *IT Professional*, 7(4):32–38, 2005.

[103] Ingo Rammer. *Advanced .NET Remoting*. APress, 2002.

[104] Donald J. Reifer. Seven hot outsourcing practices. *IEEE Software*, 21(1):14–16, 2004.

[105] Karen J. Richter. Cmmi for acquisition (cmmi-acq) primer, version 1.2. Technical Report CMU/SEI-2008-TR-010, Software Engineering Institute, Carnegie Mellon University, 2008.

[106] Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley-Blackwell, 2002.

[107] Jakub Rudzki and Tarja Systä. Small steps approach to tackling software quality in a commercial setting. *Computer Software and Applications Conference, Annual International*, 0:496–498, 2008.

[108] Michel Ruffin and Christof Ebert. Using open source software in product development: A primer. *IEEE Softw.*, 21(1):82–86, 2004.

[109] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, 2009.

[110] Hossein Saiedian and Natsu Carr. Characterizing a software process maturity model for small organizations. *SIGICE Bull.*, 23(1):2–11, 1997.

[111] Ken Schwaber. Scrum development process. In J. Sutherland and et al., editors, *OOPSLA Business Object Design and Implementation Workshop*. Springer: London., 1997.

[112] Richard W. Selby, Adam A. Porter, Douglas C. Schmidt, and Jim Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *ICSE*, pages 288–298, 1991.

[113] Petri Selonen and Jianli Xu. Validating uml models against architectural profiles. In *Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 58–67. ACM Press, September 2003.

[114] Raphaël Semeteys. Method for qualification and selection of open source software. `http://www.osbr.ca/ojs/index.php/osbr/article/view/583/540`, May 2008. Open Source Business Resource.

[115] IEEE Computer Society. Ieee std 1061-1998, ieee standard for a software quality metrics methodology. `http://ieeexplore.ieee.org/servlet/opac?punumber=6061`, dec. 1998.

[116] Software Engineering Institute, Carnegie Mellon University. Capability maturity model for software (CMM). http://www.sei.cmu.edu/cmm/, 2007.

[117] Software Engineering Institute, Carnegie Mellon University. Capability maturity model integration (CMMI). http://www.sei.cmu.edu/cmmi/index.html, 2007.

[118] So Young Sohn and Min Seok Mok. A strategic analysis for successful open source software utilization based on a structural equation model. *J. Syst. Softw.*, 81(6):1014–1024, 2008.

[119] Diomidis Spinellis and Clemens Szyperski. Guest editors' introduction: How is open source affecting software development? *IEEE Software*, 21:28–33, 2004.

[120] Mark Staples, Mahmood Niazi, Ross Jeffery, Alan Abrahams, Paul Byatt, and Russell Murphy. An exploratory study of why organizations do not adopt cmmi. *J. Syst. Softw.*, 80(6):883–895, 2007.

[121] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 274a, Washington, DC, USA, 2007. IEEE Computer Society.

[122] Ossi Syd. Avoimen lähdekoodin prosessikoneiden vertailu (comparison of open source workflow management systems). Master's thesis, Helsinki University of Technology, September 2008. in Finnish.

[123] Lars Taxén. An integration centric approach for the coordination of distributed software development projects. *Information and Software Technology*, 48(9):767 – 780, 2006. Special Issue Section: Distributed Software Development.

[124] CMMI Product Team. Cmmi for development, version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute (SEI), Carnegie Mellon University, August 2006.

[125] Chouki Tibermacine, Regis Fleurquin, and Salah Sadou. Preserving architectural choices throughout the component-based software development process. pages 121–130, 2005.

[126] David Trowbridge, Dave Mancini, Dave Quick, Gregor Hohpe, James Newkirk, and David Lavigne. *Enterprise Solution Patterns Using Microsoft .NET*. Microsoft Corporation, 2003.

[127] Aman ul haq and Muhammad Ali Babar. Tool support for automating architectural knowledge extraction. In *SHARK '09: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, pages 49–56, Washington, DC, USA, 2009. IEEE Computer Society.

[128] Kris Ven and Jan Verelst. The importance of external support in the adoption of open source server software. In *Open Source Ecosystems: Diverse Communities Interacting*, volume 299 of *IFIP Advances in Information and Communication Technology*, pages 116–128. Springer Boston, 2009.

[129] Kris Ven, Jan Verelst, and Herwig Mannaert. Should you adopt open source software? *IEEE Softw.*, 25(3):54–59, 2008.

[130] Christiane Gresse von Wangenheim, Alessandra Anacleto, and Clenio F. Salviano. Helping small companies assess software processes. *IEEE Softw.*, 23(1):91–98, 2006.

[131] Christiane Gresse von Wangenheim, Alessandra Anacleto, and Clłnio F. Salviano. Mares - a method for software process assessment in small software companies. Technical Report LQPS001.04E, LQPS - Laboratrio de Qualidade e Produtividade de Software, UNIVALI, 2004. `http://www.inf.ufsc.br/~gresse/download/LQPS001_04E.pdf`.

[132] Darja Šmite. A case study: Coordination practices in global software development. In *Product Focused Software Process Improvement*, volume 3547 of *Lecture Notes in Computer Science*, pages 234–244. Springer Berlin / Heidelberg, 2005.

[133] Željko Obrenović and Dragan Gašević. Open source software: All you do is put it together. *IEEE Software*, 24:86–95, 2007.

[134] David A. Wheeler. How to evaluate open source software / free software (oss/fs) programs. `http://www.dwheeler.com/oss_fs_eval.html`. Revised as of January 8, 2010.

[135] Pornpit Wongthongtham, Elizabeth Chang, and Tharam Dillon. Multi-site distributed software development: Issues, solutions, and challenges. In *Computational Science and Its Applications ICCSA 2007*, volume 4706 of *Lecture Notes in Computer Science*, pages 346–359. Springer Berlin / Heidelberg, August 2007.

[136] Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, Inc, 2003.

# Included Publications

Due to the copyright limitation for electronic distribution, some of the papers included in the printed version cannot be included in the electronic version. The excluded publications can be found at the publisher's sites.

[P1] Jakub Rudzki, Imed Hammouda, Tuomas Mikkola, Karri Mustonen, and Tarja Systä. Considering Subcontractors in Distributed Scrum Teams. A chapter in book Darja Smite, Nils Brede Moe, Pär J.Ågerfalk, (Eds.) 'Agility Across Time and Space: Implementing Agile Methods in Global Software Projects', p. 235-255, May 2010, Springer 2010. http://www.springerlink.com/content/u028338207130810/

[P2] Jakub Rudzki, Imed Hammouda, and Tuomas Mikkola. Agile Experiences in a Software Service Company. In Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, pp.224-228, 27-29 Aug. 2009, IEEE Computer Society. http://doi.ieeecomputersociety.org/10.1109/SEAA.2009.31

[P3] Jakub Rudzki, Kimmo Kiviluoma, Tero Poikonen, and Imed Hammouda. Evaluating Quality of Open Source Components for Reuse-Intensive Commercial Solutions. In Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, pp.11-19, 27-29 Aug. 2009, IEEE Computer Society. http://doi.ieeecomputersociety.org/10.1109/SEAA.2009.30

[P4] Jakub Rudzki, Tarja Systä, and Karri Mustonen. Subcontracting Processes in Software Service Organisations - An Experience Report. In Proceedings of the International Conference on Software Process, ICSP 2009, pp. 224-235, Springer-Verlag. http://www.springerlink.com/content/u248l6477250nk30/

[P5] Jakub Rudzki, Imed Hammouda, and Tommi Mikkonen. Ensuring Architecture Conventions in Multi-site Development. In Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008, pp.339-346, 2008. http://doi.ieeecomputersociety.org/10.1109/COMPSAC.2008.38

[P6] Jakub Rudzki and Tarja Systä. Performance Implications of Design Pattern Usage in Distributed Applications: Case Studies in J2EE and .NET. In Proceedings of the ISSTA 2006 workshop on Role of Software Architecture for Testing and Analysis, ROSATEA 2006, pp. 1-11, ACM. http://doi.acm.org/10.1145/1147249.1147250

[P7] Jakub Rudzki, Imed Hammouda, and Tommi Mikkonen. Tool Support for Quality-driven Design. In Proceedings of the 3rd Nordic Workshop on UML and Software Modeling, NWUML 2005, pp. 193-207, 2005. http://www.cs.uta.fi/reports/pdf/A-2005-3.pdf

[P8] Jakub Rudzki. How Design Patterns Affect Application Performance - a Case of a Multi-tier J2EE Application. In Proceedings of the 4th International Workshop on Scientific Engineering of Distributed Java Applications, FIDJI 2004, pp. 12-23, Springer. http://www.springerlink.com/content/f4780588t3372306/