SCIENCE • TECHNOLOGY • RESEARCH HIGHLIGHTS • VISIONS

Dissertation
6

Global software engineering

Challenges and solutions framework

Päivi Parviainen

www.r

VTT SCIENCE 6

Global software engineering

Challenges and solutions framework

Päivi Parviainen

Thesis for the degree of Doctor of Philosophy to be presented with due permission for public examination and criticism in auditorium IT116, at University of Oulu, Linnanmaa, on the 25th of May, 2012, at 12 noon.

ISBN 978-951-38-7459-9 (soft back ed.)
ISSN 2242-119X (soft back ed.)

ISBN 978-951-38-7460-5 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT
PL 1000 (Vuorimiehentie 5, Espoo)
02044 VTT
Puh. 020 722 111, faksi 020 722 4374

VTT
PB 1000 (Bergsmansvägen 5, Esbo)
FI-2044 VTT
Tfn. +358 20 722 111, telefax +358 20 722 4374

VTT Technical Research Centre of Finland
P.O. Box 1000 (Vuorimiehentie 5, Espoo)
FI-02044 VTT, Finland
Tel. +358 20 722 111, fax + 358 20 722 4374

Kopijyvä Oy, Kuopio 2012

Global software engineering
Challenges and solutions framework

[Globaali ohjelmistokehitys. Kehikko haasteista ja ratkaisuista]
Päivi Parviainen. Espoo 2012. VTT Science 6. 106 p. + app. 150 p.

Abstract

The increasingly complex and competitive market situation has resulted in Global Software Engineering (GSE) becoming more and more common practice. Companies need to use their existing resources as effectively as possible. In addition, they need to employ resources on a global scale from different sites within the company and even from partner companies throughout the world, in order to produce software at a competitive level. Thus, the ability to collaborate effectively has become a critical factor in today's software development. The main expected benefits from GSE are improvements in development time, being closer to the customers and having flexible access to better specialized and less costly resources. In practice, however, the productivity in distributed software development drops up to 50 per cent compared to single site software development. Main reasons behind this productivity drop are misunderstood or mismatched processes between teams, and poor visibility into and control of the development activities at all sites involved. The purpose of this thesis is to analyse in more detail why this is the case and what could be done to improve the situation in practice in the companies' daily work.

In this thesis, the challenges in GSE are discussed based on their root causes and then summarised into the GSE framework. The root causes are time difference and distance, multiple partners, lack of communication, coordination breakdown, different backgrounds, and lack of teamness and trust. Then solutions for these challenges are discussed from people, process and technology viewpoints and summarised into the GSE framework. As a more detailed example of challenges to a subprocess, requirements engineering (RE) in GSE is presented. RE is discussed similarly as the GSE in general, first challenges are discussed and then solutions to the challenges are presented.

The work reported in this thesis is based on extensive empirical work, carried out over several years. The empirical work was carried out in several phases: in the first phase, an industrial inventory was made, including industrial experience reported in the literature. Based on this, an initial framework for GSE was developed, consisting of the main challenges to be addressed in GSE projects. After this first phase, two sets of industrial cases were carried out, addressing a wide set of GSE aspects and challenges by trying out the GSE solutions to challenges identified in companies and validating the GSE framework. Altogether, 52 industrial cases relating to distributed development were carried out during the projects over the years 2004–2011.

This thesis shows that although GSE is common, it is still challenging and companies should carefully weigh the benefits and costs of doing the work in distributed setting vs. doing it single site. This thesis is a step towards better, more productive and higher quality GSE, as it helps companies to be aware and address potential challenges early via the GSE framework. The work presented also helps companies to find validated solutions to address the challenges in their practice.

Keywords global software engineering, requirements engineering, best practices, industrial case studies, ICT

Globaali ohjelmistokehitys
Kehikko haasteista ja ratkaisuista

[Global software engineering. Challenges and solutions framework]
Päivi Parviainen. Espoo 2012. VTT Science 6. 106 s. + liitt. 150 s.

Tiivistelmä

Jatkuva tuotteiden monimutkaistuminen ja kiihtyvä kilpailutilanne ovat johtaneet siihen, että globaali ohjelmistokehitys (GSE) on yhä yleisempää. Yritysten täytyy hyödyntää mahdollisimman tehokkaasti sekä omia että globaaleja resursseja ympäri maailman ollakseen kilpailukykyisiä. Globaalin ohjelmistokehityksen potentiaalisia hyötyjä ovat lyhyemmät tuotekehitysajat, läheisyys asiakkaan kanssa sekä mahdollisuus käyttää erikoistuneita ja edullisempia resursseja joustavasti. Käytännössä hajautetun ohjelmistokehityksen tuottavuus kuitenkin laskee jopa 50 prosenttia verrattuna paikalliseen ohjelmistokehitykseen. Tämä johtuu mm. väärinymmärretyistä tai yhteensopimattomista prosesseista tiimien välillä sekä eri paikkakunnilla tehtävän kehityksen hallitsemattomuudesta. Tutkimuksen tarkoitus on selvittää tarkemmin, miksi näin tapahtuu ja mitä voitaisiin tehdä käytännössä tilanteen parantamiseksi yritysten päivittäisessä toiminnassa.

Tässä työssä esitetään globaalin ohjelmistokehityksen haasteita sekä niiden aiheuttajia ja ratkaisuja. Haasteet esitetään osana globaalin ohjelmistokehityksen kehikkoa. Haasteita aiheuttavat aikaero ja etäisyys, useat osapuolet, kommunikoinnin puute, hallinnan hajautuminen, erilaiset taustat sekä tiimiyden ja luottamuksen menetys. Tutkimuksessa myös esitetään ratkaisuja näihin haasteisiin ihmisten, prosessin ja teknologian näkökulmasta, ja myös ne liitetään mukaan globaalin ohjelmistokehityksen kehikkoon. Tarkempana esimerkkinä GSE:n vaikutuksista osaprosessin näkökulmasta esitetään vaatimusmäärittely ja -hallinta globaalissa ohjelmistokehityksessä. Vaatimusmäärittely ja -hallinta esitetään samalla tavalla kuin globaali ohjelmistokehitys: ensin esitetään haasteita ja sitten ratkaisuja näihin haasteisiin.

Tutkimus perustuu laajaan empiiriseen aineistoon, jota on koottu usean vuoden aikana ja useassa vaiheessa. Ensimmäisessä vaiheessa tehtiin yritysten GSE-käytäntöjen nykytilan selvitys, joka sisältää kirjallisuudessa raportoidut yritysten kokemukset. Tämän perusteella laadittiin ensimmäinen versio globaalin ohjelmistokehityksen kehikosta. Kehikko sisälsi päähaasteet, jotka tulee ottaa huomioon globaaleissa ohjelmistokehitysprojekteissa. Ensimmäisen vaiheen jälkeen vietiin läpi kaksi joukkoa teollisia tapaustutkimuksia. Nämä tutkimukset kohdistuivat laajaan joukkoon globaalin ohjelmistokehityksen asioita ja haasteita. Tapaustutkimuksissa kokeiltiin ratkaisuja yrityksissä tunnistettuihin haasteisiin ja samalla validoitiin globaalin ohjelmistokehityksen kehikkoa. Yhteensä vietiin läpi 52 teollista tapaustutkimusta vuosien 2004–2011 aikana useassa eri projektissa.

Tämä tutkimus osoittaa, että vaikka globaali ohjelmistokehitys on yleistä, se on edelleen haastavaa ja yritysten täytyy huolellisesti punnita sen mahdollisia hyötyjä ja kustannuksia verrattuna paikalliseen kehittämiseen. Tämä tutkimus on askel kohti parempaa, tuottavampaa ja laadukkaampaa globaalia ohjelmistokehitystä, sillä se auttaa yrityksiä huomaamaan mahdollisia ongelmia ja reagoimaan niihin aikaisin käyttämällä globaalin ohjelmistokehityksen kehikkoa. Tutkimus myös auttaa yrityksiä löytämään hyviä ja kokeiltuja ratkaisuja käytännössä kohtaamiinsa ongelmiin.

Avainsanat global software engineering, requirements engineering, best practices, industrial case studies, ICT

Preface

This research was carried out at VTT mainly as part of the European research projects MERLIN (Embedded Systems Engineering in Collaboration) and PRISMA (Productivity in Collaborative Systems Development) during 2004–2011.

Many people have helped me during this process. First of all I mention my supervisor Professor Markku Oivo. Markku already became my thesis supervisor in the 1996, when there was some vague idea about a thesis related to measurements. Over time, the topic has matured into Global Software Engineering, and now the work is ready. I wish to thank Markku for helping to define the scope of the thesis, encouragement during the writing, and sometimes even a light push forward. Without his support this work would not have been possible.

I would also like to thank the reviewers of the thesis, Professor Kai Koskimies and Professor Stefan Biffl for their valuable comments. I'm also grateful for VTT for giving me the opportunity to work on such an interesting topic and to complete my thesis in research projects. I would also like to thank Tekes for funding these research projects.

Over the years, I've worked with many great people, both at VTT and in the companies participating in the projects, many thanks to you all! Special thanks go to the core project team of Merlin and Prisma projects at VTT, especially Maarit Tihinen. Together we struggled through many academic publication processes, motivating each other for "yet another revision" based on review comments. Maarit, I hope I can return the favour with your thesis process! I'm also grateful to my parents and family for their continuous support and encouragement during my whole life.

Finally, I would like to thank Rob, my husband, for all his support. Your support has been invaluable both outside the academic world as well as with the thesis itself; talking with you has helped to make sense of things like nothing else. You mean more to me than words can ever express, I am so happy you are in my life.

Espoo, April 2012
Päivi Parviainen

Academic dissertation

Supervisor
Professor Markku Oivo
University of Oulu
Department of Information Processing Science
P.O. Box 3000, 90014 University of Oulu, Finland

Reviewers
Professor Kai Koskimies
Tampere University of Technology,
Department of Software Systems
Box 553, FIN-33101 Tampere

Professor Stefan Biffl
Institut für Softwaretechnik und Interaktive Systeme
Technische Universität Wien
Favoritenstr. 9/188, A-1040 Wien Austria

Opponent
Professor June Verner
School of Computing and Mathematics, Keele University
Staffordshire ST5 5BG,
United Kingdom

List of papers

This thesis is based on the following original papers which are referred to in the text as I–VIII. The publications are reproduced with kind permission from the publishers.

I Hyysalo, J., Parviainen, P. & Tihinen, M. (Alphabetical order.) Collaborative embedded systems development: Survey of state of the practice. In: Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 27–30 March, 2006, Potsdam, Germany.

II Kommeren, R. & Parviainen, P. Philips experiences of global distributed software development. Empirical Software Engineering Journal, 12(6), 2007, pp. 647–660.

III Parviainen, P., Eskeli, J., Kynkäänniemi, T. & Tihinen, M. Merlin collaboration handbook: The challenges and solutions in global collaborative product development. In: Proceedings of the Third International Conference on Software and Data Technologies. Porto, Portugal, 5–8 July, 2008. Special Session on Global Software Development: Challenges and Advances on ICSOFT 2008. INSTICC. Pp. 339–346.

IV Parviainen, P. & Tihinen, M. Knowledge related challenges and solutions in GSD: Expert systems. The Journal of Knowledge Engineering, Wiley-Blackwell. Article first published online: 28 June, 2011. DOI: 10.1111/j.1468-0394.2011.00608.x.

V Parviainen, P., Tihinen, M., van Solingen, R. & Lormans, M. Requirements engineering: Dealing with the complexity of sociotechnical systems development. Chapter 1. In: Silva, A. & Mate, J. (Eds.). Requirements Engineering for Sociotechnical Systems. Information Science / IGI Global, 2005. Pp. 1–20.

VI Parviainen, P. & Tihinen, M. A survey of existing requirements engineering technologies and their coverage. International Journal of Software Engineering and Knowledge Engineering (IJSEKE), 17(6), 2007, pp. 1–24.

VII Heck, P. & Parviainen, P. Experiences on analysis of requirements quality. In: Proceedings of the Third International Conference on Software Engineering Advances. ICSEA 2008. Sliema Malta, 26–31 October, 2008. IEEE computer society. Pp. 367–372.

VIII Pesola, J.-P., Eskeli, J., Parviainen, P., Kommeren, R. & Gramza, M. Experiences of tool integration: Development and alidation. In: Mertins K., Ruggaber R., Popplewell K. & Xu X. (Eds). Enterprise Interoperability III – New Challenges and Industrial Approaches. International Conference on Interoperability of Enterprise, Software and Applications. Berlin, German. 25–28 March, 2008. Springer, 2008. Pp. 499–510.

Contents

Appendix C: Papers II and VIII, are not included in the PDF version. Please order the printed version to get the complete publication (http://www.vtt.fi/publications/index.jsp).

Terminology

Software Engineering. (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1). (IEEE, 1990.)

Software development. The development of a software product in a planned and structured process. Often used as synonymous to software engineering.

Collaboration. Product development activity that involves two or more companies, departments or customers that combine their competencies and technologies to create new shared value while, at the same time, managing their respective costs and risks. The entities can combine in any one of several different business relationships and for very different periods of time, ranging from some duration needed to exploit a particular innovation or business opportunity, to a much longer term on-going relationship. (Adapted from Welborn & Kasten 2003.)

Embedded software. Software that is part of a larger system and performs some of the requirements of that system; for example, software used in an aircraft or rapid transit system (IEEE, 1990).

Embedded software engineering. Application of engineering to embedded software, which is different than software engineering because of the specific constraints such as hard timing constraints, limited memory and power use, predefined hardware platform technology, and hardware costs (adapted from Graaf et al., 2003).

Licensor. An entity that grants the right to use a specific patented technology in return for a payment to another entity (company or individual) (adapted from Duysters & Hagedoorn, 2000).

Supplier. Supplier is the organization that provides the software work to the customer. The software work can be requirements based subcontracting, body-shopping or modifying COTS (Commercial-off-the-shelf) components or open source software) (adapted from Duysters & Hagedoorn, 2000).

Synchronous and asynchronous work. Synchronous work is work at the same time on the same data. Modifications on one shared object are carried out immediately and observed in a real time by other team members. Asynchronous work is work at different time on the same data. Modifications on shared objects are observed by other members later. (Molli et al., 2002.)

Tacit knowledge. Tacit knowledge is personal and context-specific knowledge, and it is hard to formalize and share it with others. Explicit knowledge is transmittable in formal, systematic language. Individuals occupy created knowledge and their knowledge must be made visible to others at organisation. (Adapted from Nonaka & Takeuchi 1995.)

OEM. Original Equipment Manufacturer, manufactures products or components that are purchased by a company and retailed under that purchasing company's brand name. OEM refers to the company that originally manufactured the product. (Adapted from Seppänen et al., 2001.)

1. Introduction

The increasingly complex and competitive market situation places intense demands on companies, requiring them to respond to customer needs, and to deliver more functionality and higher quality software faster. Companies need to use their existing resources as effectively as possible, and they also need to employ resources on a global scale from different sites within the company and from partner companies throughout the world. This has resulted in global software engineering (GSE) becoming increasingly common and the ability to collaborate effectively has become a critical factor in the software development life cycle. In fact, GSE of software intensive systems is the reality in most software projects: up to 80 or 90 per cent of software projects are now globally distributed and distributed development (e.g., outsourcing, one mode of distributed development) will continue to grow (Fryer & Gothe, 2008, Forrester, 2010). The current global economic slowdown has meant that efficiency and cost control are even more important for companies. Thus, organisations need to understand and evaluate globally distributed development comprehensively: not only focusing on the cost but also taking into account other aspects such as experience, reliability and continuity (i.e., the lowest hourly rate will not always give the best value in terms of investment).

The main expected benefits from GSE are improvements in development time efficiency, being close to the customers and having flexible access to greater and less costly resources. However, there are a number of problems that still remain to be solved before the full potential of GSE can be reached. One of the main problems is that development technologies are insufficiently prepared for distributed development. This leads to productivity in a distributed project dropping by as much as 50 per cent, with rework two to five times greater than for a collocated project (Fryer & Gothe, 2008; Paper II).

Reasons behind this productivity drop are multiple and have been discussed in Paper II from a company viewpoint. For example, misunderstood or mismatched processes between teams can lead to mistakes in work transfer and thus to increased rework. Also, according to Simons (2006), a team separated by as little as 100 meters can have communication reduced by as much as 95%. Distance always plays a role in GSE, but other conditions complicate the situation even more. Cultural issues, such as language barriers and differences in working or communication styles, can cause delays and affect working relationships. Visibility into and

control of the development activities at all sites can be challenging, especially when collaborating with different time zones. In addition, political issues both within the company and externally in the country or region can lead to hidden agendas and conflicting goals. For example, there may be a fear of losing jobs to the other sites. Also, the collaborating organisations may not share the same objectives, especially when reporting through different management chains or different companies, leading to competition and mistrust between parties, for example (Fryer & Gothe, 2008; Paper I; Paper III).

1.1 Research questions and scope

As discussed, the potential benefit of GSE is large, but companies do not often gain the expected benefits. The purpose of this thesis is to analyse in more detail why this is the case and what could be done to improve the situation in practice in the companies' daily work. Research on GSE practices and improvement exists, but it mostly concerns single company experiences or studies of specific aspects of GSE. However, for practitioners, a more comprehensive picture of GSE is needed. Thus, the research questions of this thesis are as follows:

- What are the main challenges faced by companies when doing GSE? How can these challenges be categorised?
- Are there solutions to address these challenges? How can these solutions be categorised in order to support locating the suitable ones for each situation?
- What are the critical activities/subprocesses in GSE? Are there solutions that support implementing these critical subprocesses?

In order to study the critical subprocesses, requirements engineering (RE) was chosen as an example. RE is chosen as it is distributed by nature as there are usually external stakeholders (e.g., users and customers), and already complex even within a single site. Distributed development complicates RE even more, as it involves a lot of interaction between the development parties. The success of the RE process has a high impact on the success of the end product, and in distributed development, the leverage effect of multiple levels of control causes the errors in RE to have a multiplied effect in terms of the dependant activities, causing extensive delays and extra effort.

1.2 Research design

The research methods in this thesis involved a literature study to define the project's basis and gather other researchers' views on the topic, and case studies and action research in order to investigate the state of actual practice in companies.

The literature study was carried out in two phases. Firstly, to develop the basis for the work by studying what others had published about GSE, and secondly, to

find solutions to the identified challenges to try out in the cases under consideration. In the first phase a broad literature search was carried out through well-known databases such as INSPEC and IEEE Xplore. The search terms were quite wide, practically anything related to GSE, with a specific focus on practical experience related to GSE. In the second phase the search area was the same, but the terms were specific for cases; for example, if challenge was related to project management in GSE, the search terms were defined accordingly. The literature study was a continuous activity, but the broad study was carried out in two main phases: in 2004 and in 2009.

Based on the first literature study, a framework was defined to collect information from industry. The industrial inventory framework (Appendix A) was developed through the extensive literature study about GSE challenges and practices reported by others and by using CMMI (2006) as a general framework to group items. All CMMI topics were included and few additional ones (collaboration management, conditions for collaboration, and sharing information) were added. GSE related subtopics were then defined for each topic. This inventory framework formed the basis of the GSE framework; the main result of this thesis. The GSE framework includes GSE challenges that should be addressed in order succeed with GSE and indication of type of solutions that can be used to take care of the challenges. The GSE framework is intended to help industry to take into account the potential challenges beforehand in order to enable addressing them in practice. Industrial sources for the study were companies participating in the Merlin (2004–2007) project, as well as companies that had published reports on their GSE experience. The inventory was carried out by performing interviews and studying the existing material from the companies, including process descriptions, templates, guidelines etc. Material and interviewees for the study were selected with the help of the company's main contact person, who knew the roles and responsibilities within their company. Interviews were carried out using the GSE framework (Appendix A). For each interviewed company, a more detailed and confidential report was written and reviewed by the interviewees and the company's main contact person. A total of 12 interviews with senior managers, project managers, software developers and testers from five different companies were carried out. In addition, two of the Merlin industrial partners filled in a questionnaire (Appendix B) instead of undertaking interviews. The questionnaire was also filled in for the five interviewed companies, resulting in a total of seven company responses. The companies represented several divergent embedded software business areas: mobile and wireless systems, embedded data-management solutions, telecommunications, embedded SW subcontracting, and consumer electronics.

Topics addressed in general included:

- Management practices, including issues such as collaboration strategy, contracts, project management, risk management, collaboration management and quality management.

- Engineering practices, including issues such as requirements development, requirements management, architecture design, software design, software implementation, integration, testing and maintenance.
- Support practices, including issues such as configuration management, change management, quality assurance, documentation, improvement processes, human-resource management, infrastructure and co-operative work.

Each of these topics had more detailed GSE-specific questions, to help in addressing all the relevant topics. In addition, the interviewees were asked if some important topic was not included in the interview questions. In addition to the interviews and questionnaire, 15 industrial cases from the literature were included in the industrial inventory. These cases were included in the inventory based on what was available during the survey time (spring 2005) and on the quality of the case description in the publication (a detailed description of the challenge and solutions used in the case). The inventory results are discussed in Paper I and from requirements engineering viewpoint in Paper V.

Based on the industrial inventory, the most important challenges for GSE were identified. Then solutions for these challenges were searched for and trialled in industrial practice. One company experiences from GSE are presented in Paper II and two other cases in Papers VI and VIII. These challenges and solutions were then collected in a Merlin collaboration handbook (Paper III). This handbook was evaluated by several external users using a structured evaluation framework.

The next phase of the research was carried out for the Prisma (2009–2011) project. In this project, in addition to two of the same companies that participated in Merlin, seven other companies participated. In order to collect data on the state-of-the- practice in these companies, several workshops were held to define challenges that were relevant for them. Solutions for these challenges were then defined and tried out. One of the cases, related to requirements quality, is described in Paper VII. These challenges and solutions were then included in the 2nd version of the GSE framework, which is described in this thesis and which is also documented as a wiki (Prisma, 2011) due to the large amount of information involved. Parts of the framework are presented in Paper IV, from knowledge engineering viewpoint.

Figure 1 presents a summary of the research design.

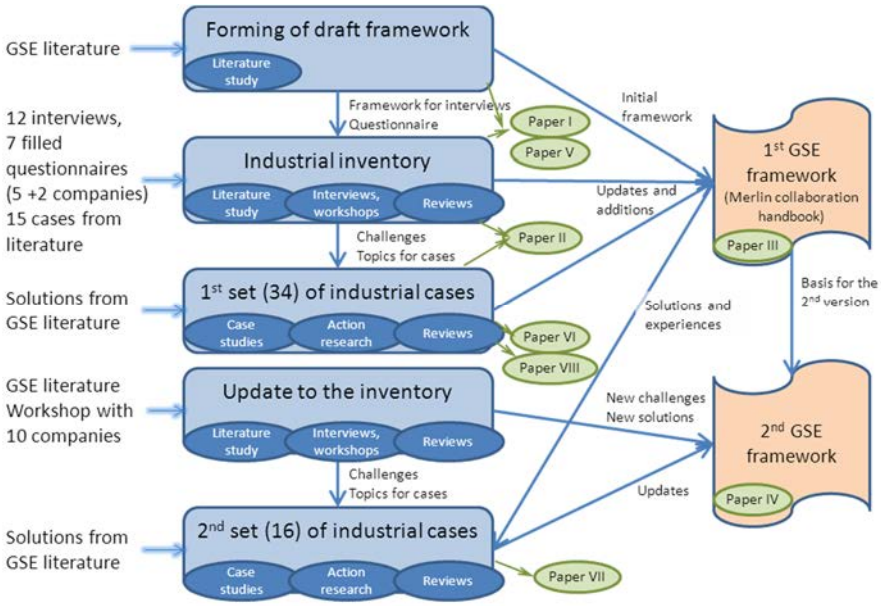


Figure 1. Research design.

During the Merlin and Prisma projects, from 2004 to 2011, a total of 54 industrial case studies were carried out. The author has actively participated in some of the cases (action research) and evaluated the results of the others (case study). Due to that the research goal was to understand the GSE and its challenges and solutions in practice, empirical work was carried out in real life industrial projects. In real life industrial projects, the ability to isolate as a basis for repetitive experiments is impossible, due to the nature of real life industrial work. As the purpose of the work was to understand the GSE challenges in industry as a whole, not some specific aspect of it, the real life industrial projects were seen as necessary.

A case study research method was used for the creation and trialling of new practices or other kinds of solutions relating to the identified challenges and problems that the companies were facing in terms of GSE. Each case study has been documented in a structured way as an experience report. Some of the empirical work was carried out as action research, as the work entailed software process improvement and technology transfer studies. According to Yin (2003), a case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and the content are not clearly evident. This method was chosen for the research as the case study methodology is well suited for many kinds of software engineering research, as the objects of study are contemporary phenomena, which are difficult to study in isolation (Easterbrook et al., 2007; Runeson & Höst, 2009). Case studies are especially appropriate for situations where the context is expected to play a role in the phenomena (e.g., if the stresses of a real project affect developers'

behaviour), or where effects are expected to be wide ranging or are expected to take a long time (e.g., weeks, months, years) to appear (Easterbrook et al., 2007) which was the case in this research.

In action research, the researchers attempt to solve a real-world problem while simultaneously studying the experience of solving the problem (Davison et al., 2004). While most empirical research methods attempt to observe the world as it currently exists, action researchers aim to intervene in the studied situations with the explicit purpose of improving the situation (Easterbrook et al., 2007). Action research is closely related to case studies: a case study is purely observational, while action research is focused on and involved in the change process. Action research consists of a five-phase cyclical process-based approach (Susman & Evered, 1978; Baskerville, 1997, Casey & Richardson, 2008) as follows: (1) diagnosing, (2) action planning, (3) action taking, (4) evaluation, and (5) specifying learning. Action research method was chosen for this research in order to give the researcher a closer look into the daily realities in a GSE project and thus to gain deeper understanding about the topic. Combination of these research methods are seen as best for this type of research, as the more detailed understanding from action research would be complemented with case study results providing a comprehensive view on the research questions.

The main new contributions of this work include:

- A comprehensive view on GSE challenges and solutions based on extensive empirical work in real industrial product development projects. Previously only scattered publications, focusing on a specific challenge or solution, or on the other hand discussing theoretical analysis of GSE challenges, has been available.
- Industrial viewpoint to GSE, i.e., challenges faced by industry have been mapped to the general GSE theory, thus making it easier for industry to find relevant solutions from the research results.
- New or enhanced solutions to challenges faced by industry in cases, that did not previously have published solutions (these are described in detail in the sameroomspirit.org wiki (Prisma, 2011)).

1.3 Outline of the thesis

Section 2 discusses GSE in general, including the benefits and risks of the GSE, GSE modes, and general challenges in GSE as presented in the literature.

Section 3 begins the empirical work and starts with industrial expressions of challenges in contracting and requirements definition, project planning and tracking, architecture analysis, design, and integration. Next, the root causes of the challenges are discussed and an example practical situation is depicted. Finally, this section presents maps the industrial challenges to the root causes.

Section 4 focuses on the solutions to the GSE challenges. The solutions are discussed from process, technology and people viewpoints. For each of these

areas the solutions are also mapped to the challenges using the GSE framework structure. Together with section 3 this section describes the GSE framework, the main result of this work.

In section 5 the chosen subprocess, requirements engineering, is discussed from GSE viewpoint. First requirements engineering is introduced in general and then the GSE impacts on the RE are discussed based on literature. Section 6 then presents the empirical work related to global requirements engineering by discussing the challenges and solutions in the globally distributed requirements engineering.

Section 7 summarises the empirical results of the work presented in the thesis by giving overview of the industrial inventory and the industrial cases carried out. For each of the cases a brief introduction, including the addressed topic from GSE viewpoint is given.

Section 8 presents the papers that this thesis summarises and extends and section 9 discusses the research results, including evaluation of the results with respect to the research goals and validity of the research. Finally, section 10 presents the summary and conclusions of the work.

2. Global software engineering

GSE means software engineering that is carried out in globally distributed settings in various geographical locations. The work can be done either within a company (multi-site development) or in collaboration between two or more companies in different locations. GSE, as used in this thesis, means product-development activity that involves two or more companies, departments or teams that combine their competencies and technologies to create new shared value while, at the same time, managing their respective costs and risks. The entities can combine in any one of several different business relationships and for very different periods of time (adapted from Welborn & Kasten 2003). An important aspect with respect to this thesis is that the work is done in distributed settings, involving sites in various geographical locations, thus incorporating asynchronous and synchronous interaction between the parties. An equally commonly used term for GSE is global software development. GSE was chosen to be used in this thesis due to the fact that the definition of software engineering is more suited to the topic of this thesis, as it covers a wider area of activities than software development. Also, the main conference of the topic uses GSE term and not GSD (ICGSE, IEEE International Conference on Global Software Engineering).

This section presents the GSE in general as presented in literature. Thus the terminology is not always unambiguous and some terms can be overlapping. This also makes clear that the GSE research is not yet mature. The terminology used in the GSE framework, main result of this work, is explained in section 3.

2.1 GSE benefits and risks

GSE has a number of potential benefits, including shortening time-to-market cycles by using time-zone differences and improving the ability to quickly respond to local customer needs. Globally distributed software engineering also allows organisations to benefit from access to a larger qualified resource pool with the promise of reduced development costs. Another potentially positive impact of globally distributed engineering is innovation, as the mix of developers with different cultural backgrounds may trigger new ideas (Ebert & De Neve, 2001; Herbsleb & Moitra, 2001; Damian et al., 2004). According to the industrial practice inventory (reported

in Paper I), the most common reasons for collaboration were to reduce development costs, to acquire competence (technology competence or knowledge of a specific market) and to avoid investing in a company's non-core competence areas. Further reasons included potential timesaving, the establishment of new business opportunities with new partners, flexibility with respect to the number of in-house resources and overcoming problems of availability of in-house resources. In some cases the company's whole business can be based on collaboration; for example, if the company is developing a component that is meant to be used as part of another product such as a COTS (commercial-off-the-shelf) product, or if it is offering human resources, i.e., developers (expertise providers and consulting companies).

There are also several risks involved in GSE. The general risks mentioned in the survey of state-of-the-practice (Paper I) had to do with the openness of communication between partners; for example, problem-hiding may be an issue in customer–supplier relations. Furthermore, unclear assignments, lack of trust between partners, difficulties in agreeing on intellectual property rights and the unreliability of the partners' development schedule were seen as risk factors for any mode of collaboration. From the supplier's or licensor's viewpoint, the risks mentioned concerned the continuation of the collaboration in the future and predicting the most saleable product features correctly during road-mapping. On the other hand, from the customer's point of view, the quality of the acquired product (e.g., reliability and performance) and becoming too dependent on one partner were seen as risks. Finally, competence issues, such as the competence of new partners and a weakening of one's own competence were also mentioned as risks. These risks are similar to those that other authors have highlighted. For example, Ebert et al. (2008) have identified a top-ten risk list for GSE projects over the past decade in a multitude of GSE projects and situations covering four continents. They are not specific to particular industries or company sizes, but rather to the underlying life-cycle processes and management practices. The risk list is as follows:

1. *Project-delivery failures,* the risk of being late or over budget amplifies in probability and impact due to the difficulties of managing a global development team.
2. *Insufficient quality,* in GSE, many work products are moved across places and teams with the risk of insufficient quality due to that, the teams suppose that there will still be sufficient validation "downstream" so that quality deficiencies accumulate.
3. *Distance and culture clashes,* GSE is highly impacted by work organization and effective work split. GSE causes overheads for planning and managing people, e.g., due to language and cultural barriers.
4. *Staff turnover (mostly for captive centres)*, is a specific risk especially in Asian countries due to abundant job opportunities in the respective economies. High turnover requires more investment in training and monitoring of the work.

5. *Poor supplier services (only for outsourced GSE)*, a frequent risk with third party suppliers is not meeting the expectations in terms of quality and delivery schedule.
6. *Instability with overly high change rates,* often being present in different markets with individual engineering teams means that each of the teams first of all looks to needs of the local market. When products and features are assembled, inconsistencies appear which cause late requirements changes.
7. *Insufficient competences*, is amplified in GSE by the bigger dependencies given the globally distributed team combined with less visibility on resource planning and skills availability.
8. *Wage and cost inflation*, the global fight for software engineering talent creates a major risk of wage inflation, causing it more expensive to keep the required competences working in the project.
9. *Lock-in (only for outsourced GSE)*, with GSE supplier competition on a global market, external suppliers often start with rather low rates and once the projects are sufficiently large clients might be forced to lock-in with them due to progress of product development and knowledge transition. In the least we may have to face increasing cost inflation.
10. *Inadequate IPR management,* software is not patented and copyrights are not enforced equally in all regions of the world. Further risks are related to improper use of external software (e.g., OSS) and careless handling of confidential information.

These risks are partly challenges as presented in section 3, and partly consequences if the challenges are not addressed well.

2.2 Global software engineering modes

There is no commonly accepted definition for GSE or collaboration modes. Instead, there are a multitude of terms that are used which mean some mode of GSE. This section will explain a few of these modes. These modes were chosen in order to give different viewpoints of the GSE modes. First, modes differentiated by the type of agreement will be presented. Next, other categorisations presented in the literature are discussed, including equity and non-equity collaborations, upstream, horizontal and downstream collaboration, dyadic alliances, alliance constellations and alliance networks and explorative and exploitative collaboration.

A way to group the modes is based on the type of agreement (i.e., the contract defining the relationship and product ownership), which can be placed into one of four main categories (adapted from Duysters & Hagedoom, 2000; Williamson, 1996; Hagedoorn, 2000):

1. In customer–supplier relationships, the customer is the organisation that is buying the software work (or technology or knowledge) from the supplier. Work may be based on requirements given or on the modification of existing COTS products or open-source code. The customer may also hire workers from a supplier in so-called body-shopping. The supplier is the organisation that provides the software work to the customer. Three main types of relationships are identified: requirements-based subcontracting, body-shopping and MOTS (modified-off-the-shelf) components.
2. With technology exchange/licensing, a company is granted the right to use a specific patented technology in return for a payment. Companies may also define open interfaces for products that allow any interested party to create software/services for the product. Types of technology exchange/licensing include: COTS components, open-source software and open architectures.
3. Joint research and development (R&D or partnering) includes either joint ventures that are organisational units created and controlled by two or more parent companies or joint development agreements that cover technology and R&D sharing between two or more companies in a joint research or joint development project.
4. In-house distributed development where development is organised within one company (legal entity), usually without contracts, but often organised over different sites, which can be located in different countries and continents.

This categorisation is used throughout this thesis, and the main focus is on customer–supplier relations and in-house distributed development, as those have been the most commonly used models in the case organisations and in the available related work.

To further describe the various modes, four different ways to distinguish the modes of collaboration between collaborating organisations are presented (Faems, 2003): equity and non-equity collaborations, upstream, horizontal and downstream collaboration, dyadic alliances, alliance constellations and alliance networks and explorative and exploitative collaboration.

Equity and non-equity collaborations. In equity arrangements each partner has an equity position and expects a proportional share of the dividend as compensation. Joint ventures and minority equity investments are examples of such arrangements. Contractual arrangements (non-equity) refer to a wide array of inter-firm linkages such as joint R&D and strategic R&D alliances.

Upstream, horizontal and downstream collaboration. Upstream, horizontal and downstream collaboration refers to the relative position of the involved organisations. Collaboration with universities, research institutes, government laboratories and suppliers are examples of upstream alliances. Alliances with clients, distribution or marketing companies are downstream examples. Horizontal collaboration embodies the alliances with competitors or complementors. This is also sometimes called

vertical and horizontal integration, where vertical integration covers upstream and downstream integration (Lindström, 2003). Lindström (2003) also discusses virtual integration where partners' businesses are joined with OEMs (original-equipment manufacturers) and the suppliers are treated as if they were inside the company.

Dyadic alliances, alliance constellations and alliance networks. In a dyadic alliance only two organisations collaborate and alliance constellations involve more than two partners. An alliance network refers to a collection of an organisation's dyadic alliances and alliance constellations. An alliance means that two or more firms create a unique organisational entity, in which each firm retains its individual identity and internal control. The purpose of an alliance is to (1) achieve joint strategic goals, (2) reduce risk while increasing rewards, and/or (3) leverage resources.

Explorative and exploitative collaboration. Explorative and exploitative collaboration can be distinguished by the collaboration objectives. For example, R&D joint ventures, research consortia and joint R&D agreements are mostly long-term strategic alliances, while technology exchange agreements or customer–supplier relationships are more often categorised as short-term cost-economising or revenue-generating alliances.

The following table shows the relationship between the first categorisation (type of agreement) of collaboration mode with the other classifications.

Table 1. Collaboration-mode classifications.

	Equity and non-equity	***Upstream, horizontal and downstream***	***Dyadic alliances, alliance constellations and alliance networks***	***Explorative and exploitative***
Customer–supplier relationships	Non-equity	All	None	Exploitative
Technology exchange/licensing	Non-equity	All	All	Exploitative
Joint R&D	Equity	All	All	Explorative
In-house distributed development	Equity	-	-	Explorative and exploitative

These different categorisations show how many different kinds of collaboration modes exist in GSE. The applicable mode depends on the situation, with one mode being more suited to a specific situation than another. For example, Lindström (2003) presents a decision matrix to help in choosing the mode (see Table 2).

Table 2. Strategic-alliance model decision-making matrix (Lindström 2003).

MARKET AND DEMAND				
	NEW AND UNFAMILIAR	Joint venture Horizontal/Virtual integration	Horizontal/Virtual integration Licensing Equity investment	Equity investment Joint venture Research consortia
	NEW BUT FAMILIAR	Supplier integration Short- and medium-term purchasing agreements	Vertical/Virtual integration Licensing	Technology partnership Standardization efforts
	KNOWN	International development Long-term purchasing agreements	Supplier integration Joint R&D Licensing	Technology exchange Technology partnership Research consortia
		EXISTING	**NEW BUT FAMILIAR**	**NEW AND UNFAMILIAR**
		TECHNOLOGY		

The commonly used GSE terms such as distributed development, multi-site development, outsourcing, off-shoring and inter-company collaboration are usually customer–supplier types of modes. Table 3 explains the specifics of these modes.

Table 3. Commonly used terms to describe GSE modes.

Mode	***Description***	***Mapping***
Distributed development	General term meaning development that happens in at least two different geographical sites.	Can include any of the collaboration modes
Multi-site development	Development that happens within one company in at least two different geographical sites. Is sometimes also used to refer to distributed development in general.	In-house distributed development
Outsourcing	Outsourcing is usually used to mean the contracting out of a business function to an external provider. Multisourcing means provisioning and blending services from the optimal set of internal and external providers in the pursuit of business goals. Outsourcing is one of the GSE modes, main distinguishing aspect being that one company is buying work or parts of product from some other company and this is specified in a contract.	Customer–supplier mode
Off-shoring, offshore outsourcing	Outsourcing to suppliers outside the nation. There is also a derivate of this called nearshoring, which means that the business has shifted work to a lower cost organisation within its region.	Customer–supplier mode

Within these modes, the work may be carried out on different interaction levels. Thompson (2001) stated that there are at least four different interaction modes:

1. Hierarchical (also the formal-process or assembly-line model). Individuals perform the tasks assigned to them in isolation from each other. In the hierarchical model, authority is usually singular.
2. Swap meet (also division of labour). Individuals perform tasks in isolation but come back together to review or revise.
3. Asymmetrical. Collaborators cannot be viewed as equal (e.g., teacher–student collaboration).
4. Dialogic (also integrative, integrated-team or symphony model). Tasks are performed together by all members of the group.

The level of required interaction depends on the type of work that is done; for example, the ability to divide work so that there is little interaction needed and consideration as to the level of know-how of the different parties. In practice, the interaction is usually a mix of these, so that some tasks are done individually and some are carried through in a more dialogic mode. In order to avoid too many dependencies between sites and partners, the work should be divided purposefully. According to Grinter et al. (1999), modes for coordinating R&D work across multiple sites include:

- Functional areas of expertise: Expertise for a specific functional area involved in development of the product is located at a single site.
- Product structure: Organisation is split among sites along lines suggested by the product architecture.
- Process steps: Work is broken up into process steps such as systems engineering and testing and these steps are used as handoffs among various locations.
- Customisation: One geographical site owns the core code for the product and other sites make changes to the code base such as adding features and enhancements for a specific customer base.

2.3 General GSE challenges

GSE challenges are discussed in many publications from various perspectives. Silva et al. (2010) carried out a systematic literature review of distributed development challenges, best practices, models and tools. They analysed 54 papers and found that the top-five challenges appeared in 120 pieces of evidence (45%) out of a total of 266 for all 30 identified challenges. These five challenges were effective communication, cultural differences, coordination, time-zone differences and trust. Similar to these findings and a commonly referenced classification for

challenges caused by globally distributed development is (Carmel, 1999; Carmel & Tija, 2005):

- Communication breakdown (loss of communication richness)
- Coordination breakdown
- Control breakdown (geographical dispersion)
- Cohesion barriers (loss of "teamness")
- Culture clash (cultural differences).

These challenges affect all aspects of product development and different authors have studied these aspects in more detail either from certain process viewpoints or from the challenge viewpoint. Next, each of these aspects is discussed in more detail based on Carmel (1999), Carmel and Tija (2005) and Paper I.

Communication breakdown (loss of communication richness). Human beings communicate best when they are communicating face-to-face. A software engineer would usually prefer to conduct a difficult design session face-to-face because people communicate with more than mere words (e.g., drawings on whiteboards, body language etc.). Communication over distance frequently leads to misinterpretation and that often leads to errors in development. In a distributed project, people cannot communicate well due to language barriers and the unavailability of resources. In addition, according to Herbsleb et al. (2001), distribution may hinder informal or unplanned communication. In distributed development, all this has to be managed and supported with tools such as groupware and by ensuring the quality of documentation.

Coordination breakdown. Software development is a complex task that requires on-going adjustments and coordination of shared tasks. In geographically distributed projects, the small adjustments usually made in face-to-face contact do not take place or it is not easy to make adjustments. Thus, problem solving gets delayed or the project goes down the wrong track until it becomes very expensive to fix. GSE also sets additional requirements for planning; for example, the need for coordination between teams and the procedures and contacts for how to work with partners need to be defined (Damian & Zowghi, 2002; Paasivaara & Lassenius, 2003; Herbsleb & Mockus, 2003). Wahyudin et al. (2007) also state that GSE demands more from project management: in addition to project manager, also the project members such as testers, technical leader, and developers also need to be kept informed and notified for certain information and events which are relevant to their roles' objectives in timely manner and provide basis for in-time decision making. Coordination breakdown can also cause a number of specific problems; for example, Battin et al. (2001) reported a number of software integration problems, which were due to a large number of independent teams.

Control breakdown (geographical dispersion). GSE means that management by walking around the development team is not feasible and, instead, telephones, E-mail and other communication means (e.g., chat servers) have to be used. This kind of communication is less effective and cannot always give a clear and correct status of the development site. Also, dividing the tasks and work across development sites, and managing the dependencies between sites is difficult

due to the restraints of the available resources, the level of expertise and the infrastructure (Herbsleb & Moitra, 2001; Welborn & Kasten, 2003; Battin et al., 2001). According to Holmstrom et al. (2006), despite flexible working hours and communication technologies that enable asynchronous communication, creating the overlap in time between different sites is challenging. Lack of overlap leads to a delay in responses with a feeling of "being behind", "missing out" and even losing track of the overall work process.

Cohesion barriers (loss of "teamness"). In working groups that are composed of dispersed individuals, the team is unlikely to form tight social bonds, which are a key to a project success. Lack of informal communication, different processes and practices have a negative impact on teamness (Damian & Zowghi, 2002; Herbsleb & Mockus, 2003; Battin et al., 2001). Casey and Richardson (2008) outlined that fear (e.g., of losing one's job to the other site) directly impacted negatively on trust, team-building co-operation and knowledge transfer, even where good relationships existed beforehand. They also stated that fear and lack of trust negatively impacted on the building of effective distributed teams, resulting in clear examples of not wanting to cooperate and share knowledge with remote colleagues. Al-Ani and Redmiles (2009) discuss the role that the existing tools can play in developing trust and providing insights on how future tools can be designed to promote trust. They found that tools can promote trust by sharing information derived from each developer's activities and their interdependencies, leading to a greater likelihood that team members will rely on each other and thus leading to a more effective collaboration. Holmstrom et al. (2006) found that establishing and maintaining teamness in a distributed development environment is difficult: while websites with photos and individual profiles serve a purpose, the common solution still seems to be travelling between sites.

Culture clash (cultural differences). Each culture has different communication norms. The result of these differences is that in any cross-cultural communication the receiver is more likely to misinterpret messages or cues. Hence, the miscommunication across cultures is always present. Borchers (2003) discusses observations of how cultural differences impacted the software engineering techniques used in the case projects. The cultural indexes, power distance (degree of inequality of managers vs. subordinates), uncertainty avoidance (tolerance for uncertainty about the future) and individualism (strength of the relationship between an individual and their societal group), discussed by Hofstede (2001), were found to be relevant from the software engineering viewpoint. Holmstrom et al. (2006) discuss the inherent challenge of creating a mutual understanding between people from different backgrounds. Often, general understanding in terms of English is considered as good, but more subtle issues, such as political or religious values, cause misunderstandings and conflicts during projects.

Cultural differences are also discussed in general (not specifically from a software engineering viewpoint) in several publications (e.g., Hofstede, 2001; Trompenaars & Hampden-Turner, 1997).

These general challenges are used (adapted) in the discussion of challenges and problems faced by the case companies in practice in the remainder of this thesis.

3. GSE challenges

The previous section focused on GSE challenges on a general level, as discussed in the GSE literature. Based on the empirical work carried out during the Merlin and Prisma projects, several concrete challenges have been identified that make GSE less productive in companies in practice. The industrial companies express the challenges differently than theory, thus the GSE framework presented in this thesis (Sections 3 and 4) is needed to help companies in finding relevant solutions to the challenges they are facing.

3.1 Industrial expressions of challenges

The most critical points in global software engineering, based on the industrial inventory and expressed by the Merlin and Prisma partners in workshops, were the contracting and requirements definition, project planning and tracking, architecture analysis and design and integration. These points are discussed next in more detail and are described in more detail in Paper I.

Contracting and requirements definition: The more detailed the prepared specification of the work is, the better (within a reasonable degree of effort). Thus, if all collaboration partners have the same view/shared understanding of what is to be done and if that is documented well, fewer conflicts will occur. Challenges and problem expressions relating to this point were:

- "My supplier delivers things I didn't ask for or it's always a surprise what I get from my supplier."
- "The integration of the parts developed by the supplier takes a lot of effort."
- "It's not clear what my assignment is (supplier viewpoint)."
- "My customer changes the assignment continuously."
- "My customer's requests are unclear and vague and not communicated."
- "We have trouble with varying interpretations of the requirements when the definition process is distributed."

- "Management of the changes made to the requirements by different parties of the project is difficult."
- "Validating each individual project stakeholder's interpretation of the requirements before the implementation takes place is difficult."
- "It's challenging to prioritise requirements correctly and effectively when some stakeholders are 'louder' than others."

Project planning and tracking: It is important to clearly define status-reporting practices and change management procedures, including the details on reporting channels, decision authorities and escalation channels. Other important issues in collaborative project planning are the identification of dependencies between partners – for example, the interdependencies of the subsystem deliveries – and taking these into account in project schedules. Challenges and problem expressions relating to this point were:

- "I don't know what is happening, what my partners are doing."
- "It is impossible for the project manager to get an overview of the status of the software."
- "I'm not sure whether the resources are used optimally."
- "Management has high difficulty in determining project status when not present on the same site as the project."
- "The suppliers' work is often delayed."
- "Reporting the progress of the project to the managers at the right level (not too much, not too little) is difficult."
- "It's challenging to communicate the 'project vision' with a clear enough scope to get a project started in the early phases."
- "We suffer from delays caused by having to wait for input from other teams."

Architecture analysis/design: Architecture is one of the key disciplines enabling successful collaboration. In particular, a lack of sound architecture leads to poor integrability. Ensuring that all partners understand the architecture correctly is difficult and the required level of communication is often underestimated. Challenges and problem expressions relating to this point were:

- "It's difficult to know when there is too little design and when is there 'analysis paralysis'."
- "The interfaces between the subsystem assigned to my team and subsystems assigned to other teams are poorly defined."
- "How can we establish a common understanding of architecture in all of the development sites?"

Integration and testing: While integration is often the most time- and effort-consuming activity even in in-house product development, GSE brings additional complexity; for example, new actors and communication requirements. Challenges and problem expressions relating to this point were:

- "The quality of supplied parts is low."
- "We have trouble with poor quality and reusability of existing code in distributed development."
- "We have problems at integration when remote programmers throw their build code 'over the wall' to a build manager who must resolve conflicts."
- "Integration of the software takes place at different locations and is poorly coordinated, finally leading to a non-buildable product."
- "It's not traceable whether the product meets the requirements in GSE."
- "It's not clear who is responsible for resolving issues (defects, mismatching parts) detected in integration."

Co-operative work: Good, that is, timely and accurate communication is regarded as a crucial success factor for projects. Openness of communication and multi-site/multi-partner culture were considered as very important for collaboration success. Challenges and problem expressions relating to this point were:

- "How can I order/arrange the information flow for a project so that someone else can also find that information later on (documents/e-mails/phone calls/etc.)?"
- "How can I communicate/filter the requirements that are in the offer/agreement to different participants in the project team?"
- "We have problems with unsolicited and ad hoc changes."
- "I would like to minimise the learning curve for my partners to learn new tools, methods and technologies. How can I do it?"
- "When moving data from the requirements'-management tool to the test tooling, defects are introduced."
- "How can I improve trust and confidence between the teams implementing different code lines?"
- "My partner uses different tools and there are difficulties transferring data from one tool to another."
- "Coordinating the information flow when subcontractors are in different time zones is difficult."
- "We have difficulties dealing with conflict between team members who are not face-to-face."

- "We have problems coordinating people across multiple sites and also across multiple projects."
- "It is not clear what modules and tests are affected when we accept this change request."

In addition to these issues, product road-mapping was also seen as one of the most important areas from the component-supplier viewpoint because taking into account and prioritising the customers' future requests is a complex task, especially when there are several customers with varying requests.

These expressions are partly challenges and partly problems when challenges have not been addressed properly. These expressions and industrial inventory were used as a basis for the GSE framework challenges that are discussed in the following sections.

3.2 Root causes of the challenges

As becomes clear from section 3.1, the companies face many problems and challenges, and their expressions of them are varying: the companies express the issues as problems or challenging areas, using different terminology, overlapping with other issues, and in a scattered way. Thus, in order to gain a more comprehensive view on the GSE challenges in general, a structure to depict the problem and challenge area is needed. The challenges discussed in section 3.1 and the other GSE challenges identified in the cases and literature experience can be related to root causes of the problems in GSE. These root causes are presented in Figure 2.

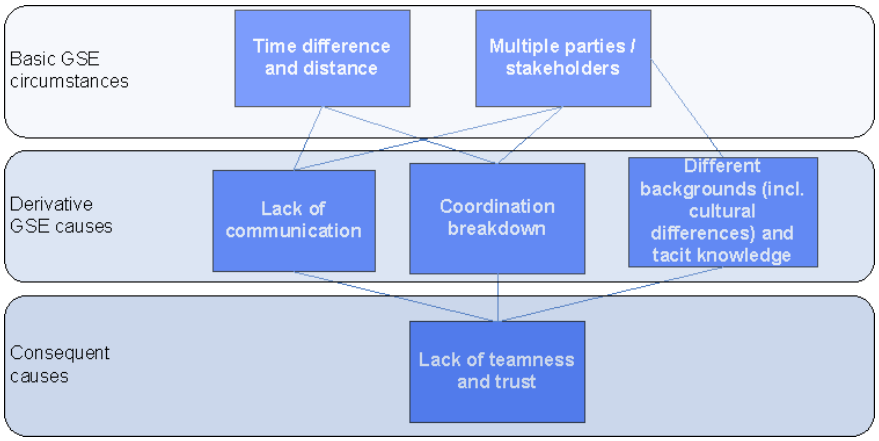


Figure 2. Root causes of problems in GSE.

The root causes were identified from various literature sources and in the empirical work. Root causes are the fundamental reasons behind the challenges in GSE, i.e., they cause challenges that can be addressed via specific solutions (e.g., practices). Some of them are not necessarily GSE specific, such as multiple parties/stakeholders, but they are causes that are present in every GSE project. Root causes are things that cause multiple challenges in GSE, challenges can be addressed by solutions and if they are not properly addressed, problems appear. These root causes are derived from two sources: 1. literature, by analysing what other researchers have proposed as causes and their relations, and 2. from challenges expressed by industry and the causes behind them.

The root causes are discussed from three aspects – people, processes and technology – in order to gain a comprehensive understanding of the causes. The root causes have been discussed in the literature review, as presented in section 2.3. However, they have been presented in various papers in a disorganised manner and have not yet been analysed systematically, as presented here. Also, they have not been connected to the industrial problems and challenges; that is, industry faces problems caused by these issues, not from these issues themselves. For example, a company does not state as a problem that there is a challenges of differences in backgrounds or coordination breakdown, but that they get bad quality deliveries from the other sites. Moreover, the root causes are interconnected and the relations are marked with lines in Figure 2. In summary, time difference and distance can lead to a lack of communication and coordination breakdown, multiple partners may cause a lack of communication and coordination breakdown, and the people involved have different backgrounds and, thus, different tacit knowledge. These causes together – if not properly addressed – lead to a lack of "teamness" and trust. Next, each of these root causes is discussed in more detail (B1, B2, D1, D2, D3, C1 and C2 are identifications of the causes used as references in the remainder of the thesis).

The **basic GSE circumstances** include matters that are an intrinsic natural part of GSE. They directly complicate GSE as well as cause further challenges. The basic circumstances are:

- Multiple parties (B1), meaning two or more different teams and sites (locations) of a company or different companies. When multiple parties are involved, different working cultures and backgrounds usually play a role.
- Time difference and distance (B2) are caused by the geographical distribution of the parties. Distance is always present in GSE and the extent of the distance seems to be less relevant, as research has shown that a team separated by as little as 100 meters can have communication reduced by as much as 95% (Simons, 2006). However, time difference may not always be present in distributed development if the parties are in the same or in nearby time zones.

Example problems that are directly caused by these basic circumstances include issues such as unclear roles and responsibilities for the different stakeholders in

different parties/locations, knowing the contact persons (responsibilities, authorities and knowledge) from different locations and establishing and ensuring a common understanding across distance. The basic circumstances can also cause poor transparency and control of remote activities, difficulties in managing dependencies over distance, problems in coordination and control of the distributed work and integration problems, for example. Basic circumstances may also cause problems in terms of accessing remote databases and tools, they may generate data-transfer problems caused by the various data formats between the tools or different versions of the tools used by the different teams and the basic circumstances may also cause problems with data security and access to databases or another organisation's resources. Although these issues are directly caused by the basic circumstances, they may be amplified by the derivative causes.

Derivative GSE causes (see Figure 2) are causes that may not be present in every GSE situation and their effects can often be avoided with proper practices and ways of working. The derivative causes are:

- Lack of communication (D1): Communication is difficult in geographically distributed development. For example, if there is a lack of overlapping working hours due to the time-zone differences, arranging face-to-face meetings is complicated and expensive. Distribution may also hinder informal or unplanned communication as all this has to be managed and supported through communication tools, and still the richness of communication (e.g., due to body language not being visible) may suffer. The loss of communication richness also creates miscommunication. Additionally, the problems in distributed development are not solved as effectively as in co-located development because of the communication-related issues.
- Coordination breakdown (D2): Dividing and coordinating the tasks and work across development sites is difficult due to the restraints in the available resources, differences in levels of expertise and infrastructure, for example.
- Different backgrounds (D3) may imply that the ways in which teams work are not the same as are assumed and this can cause problems. Different backgrounds also involve differences in tacit knowledge causing misunderstandings and wrong assumptions.

Example problems coming from these causes include, for example, ineffective use of resources as competences are not known from other sites, obstacles in resolving seemingly small problems and faulty work products due to a lack of competence or background information. These causes can also lead to a lack of transparency in the other parties' work, misunderstood assignments and, thus, faulty deliveries from parties, delays caused by waiting for the other parties' input and duplicate work or uncovered areas. Further problems that can be caused by these issues include differences in tool use or practices in storing information, misplaced restrictions on the access to data and unsuitable infrastructure for the distributed

setting. If these issues are not addressed well, the appearance of consequent problems is more likely.

Consequent causes are causes that are not always present in GSE, but if they are, they have a major impact on the distributed software development project. The basic circumstances and derivative causes increase the chance of these problems being present. By addressing the derivative causes well, these problems may be avoided. Consequent causes include:

- Lack of teamness (C1) refers to a lack in the creation and maintenance of a common bond and identity in a team. Teamness helps a team to work better together as it improves co-operation and commitment to the team's goals. Different processes, practices and cultures tend to diminish teamness.
- Lack of trust (C2) refers to mistrusting partners, manifesting as an unwillingness to help each other and the placing of blame instead of working together towards a common target. The role of trust is always significant in collaboration as it is very difficult, if not impossible, to make "perfect" contracts, covering all aspects of a relationship.

Example problems coming from these causes include hiding problems and an unwillingness to ask for clarification from others, expending a lot of effort in trying to find that the cause of problems (defects) has occurred in the other parties' workplace, an unwillingness to help others and an unwillingness to share information and work products until specifically requested to do so. These causes may also appear as difficulties in agreeing about the practices to be used and then not following the process and practices as agreed, for example. Further problems caused by these issues include the use of other tools than those agreed to for the project and plentiful technical issues that hinder communication and use of the tools, as agreed.

3.3 Example situation to highlight challenges

A simplified situation of globally distributed product development is presented as an example of the challenges in Figure 3. In this situation, company 1 is responsible for the product development and has subcontracted part of the development to other companies: companies 2 and 3 are software suppliers and company 4 provides system-testing services. Company 1 defines the requirements and architecture of the product, makes some part of the software in-house, integrates the product and releases the product once testing results are satisfactory. Each of the companies is in a different location and company 1 has several teams in-house working on the product.

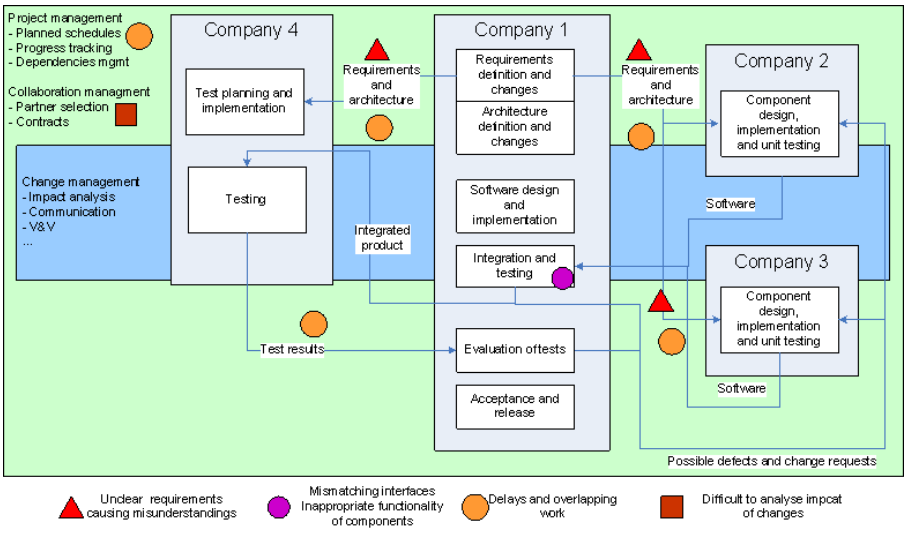


Figure 3. Example GSE situation and some of its challenges.

In the example situation, several problems are likely to appear. For example, threats are included in the figure and are explained next.

Unclear requirements causing misunderstandings: Requirements are the basis (main input) for work for the component developers and for the system testers. If requirements are not stated clearly and unambiguously, parties may make their own interpretations of the requirements. This, in turn, may result in poor design decisions and may lead to a delay in the integration phase of the project. The effect of unambiguous and changing requirements is higher in GSE because of the leverage effect caused by the multiple levels of control (errors may be repeated on various levels and with different involved parties) and information transfer. The misunderstandings may be found only during integration when the pieces from different parties are put together. Basically, this problem is caused by the fact that there are multiple parties (B1) in different locations (B2) with different backgrounds (D3) making different interpretations and assumptions about the requirements. There may also not have been enough communication (D1) about the requirements.

Mismatching interfaces and inappropriate functionality of components: Mismatching interfaces will cause difficulties in integration, as parts developed by different parties will not work together as intended. Also, fixing the problems is difficult and laborious, as tracking the source of problems from parts made by various parties is difficult. Inappropriate functionality can mean missing, duplicate or unnecessary functionality of the software, leading to a product not meeting its requirements. This problem is caused by the basic circumstances (B1 and B2), and by different backgrounds (D3) (different ways of working), coordination breakdown (D2) dividing the responsibility of product development and by a lack of

communication (D1) of the requirements, architecture and intermediate work products.

Delays and overlapping work: In distributed settings it is more difficult to keep track of other partners' work due to a lack of insight into the partners' work. Keeping track is important in order to enable a fast reaction to and prevention of problems during development. Also, dependencies between partners may not be obvious and may need to be managed in order to prevent parties needing to wait for each other. Overlapping work may come about if the requirements and architecture are not defined clearly and unambiguously. Moreover, not enough communication may have taken place so as to enable one partner to view how the other partner's work is developing. This problem is caused by the basic circumstances (B1, B2), and the derivative causes of coordination breakdown (D2) and a lack of communication (D1).

Incorrect analysis of the impact of changes: Analysing the impact of changes is more difficult in distributed development, as the work of others may not be known in much detail and there is a chance of an extended impact. Incorrect impact analysis leads to increased effort and frustration when bouncing change requests back and forth as well as to the potential for mismatches in functionality of the parts developed by different parties. This challenge is caused by the basic circumstances (B1, B2) and the derivative causes of different backgrounds (D3), when all partners do not have the same understanding of the product and may make wrong assumptions of the impact of the change. It is also caused by coordination breakdown (D2), as it might not be so clear as to what the other partners were doing and what dependencies they have.

These were just a few examples of challenges and problems illustrated in a simplified case. Many others, such as suboptimal use of resources, perceived low quality of supplied parts and sharing the correct level of information (right time, right amount to right people) may also occur.

3.4 GSE challenges framework

This chapter summarises the discussion of the GSE challenges and problems presented in the previous chapters. The structure used to present the challenges and problems is the same as the industrial inventory framework (Appendix A), and is based on CMMI, complemented with some GSE practices. The complementary GSE practices are collaboration management and co-operative work, and also each CMMI subarea has been described from GSE viewpoint. The summary of challenges is meant to help in identifying the challenges that are relevant for a given situation. The table is based on the discussion in Sections 3.1 and 3.2, bringing together the industrial expressions of challenges (the industrial inventory, industrial cases and workshops) as well as the discussion of challenges caused by the root causes. The table structure is based on literature and was also used as the industrial inventory framework.

Table 4. GSE challenges' summary.

	Basic GSE circumstances - Multiple parties/ stakeholders - Time difference and distance	**Derivative GSE causes** - Lack of communication - Coordination breakdown - Different backgrounds	**Consequent causes** - Lack of teamness and trust
Management practices			
Collaboration strategy	Ineffective collaboration due to unsuitable collaboration mode	Delays and problems in co-operation due to ad hoc way of working	Lack of collaborative work culture due to not paying attention to collaborative work requirements
Contracting practices	Unclear or non-existent agreements between partners	Gaps and/or duplicate items in contracts coverage	Lengthy and troublesome contracting process
Conditions for collaboration	Delays and wasted effort due to inappropriate project organisation or missing collaborative practices and tools	Lack of visibility of partners' work	Unwillingness to collaborate and help each other. Mistrust of each other's work results
Supplier management	Gaps or duplicate work caused by unclear assignments, delays caused by unmanaged dependencies	No or not enough visibility of suppliers' work. Unexpected deliveries (content or timing). Misunderstood requirements	Supplier not receiving the support (e.g., input, feedback) needed. No openness to share problems or ask questions
Project management	Unclear status of project. Delays caused by unclear decision-making authorities and practices	Delays caused by unaligned teams. Mismatching work results caused by misunderstood goals. Unavailable resources	Mistakes due to wasted effort caused by missing information. Wasted time caused by checking each other's work needlessly

Engineering practices			
Requirements engineering	Difficulties in prioritising requirements due to various views, unclear requirement documentation, difficulties in establishing and managing traceability	Misunderstood requirements causing unfitting deliveries. Problems caused by non-communicated changes to requirements. Inconsistencies between requirements and further work products	Lack of clarity in requirements that are not communicated but where (wrong) assumptions are made
Architecture	Unsuitable architecture (not supporting distributed development)	Unclear, non-communicated or misunderstood architecture	Architecture and architectural rule violations
Design and coding	Difficulties in communicating design rationale causing mistakes in design decisions	Different coding styles and standards affecting understandability of design and code by others	Mistrust towards design and code made by others causing unnecessary effort in checking the work
Integration	Delays in responses for issues found in integration, unavailability of information	Problems during integration caused by mismatching pieces or unsynchronised deliveries	Delays in integration due to lack of commitment from implementers (code is ready only when it is integrated into the product)
Testing	Difficulties in replicating defects found in testing, lack of visibility in testing the progress	Duplicate testing efforts, inadequate testing (due to lack of knowledge), ineffective testing caused by unsynchronised deliveries	Blaming other parties for the defect sources rather than constructively finding out the source of the defect, taking test results personally
Supporting practices			
Configuration management	Complex configuration-management practices. Difficulties in finding correct versions	Use of wrong versions, problems in builds, unclear status of development work	Unwillingness to follow defined rules and use the tools as agreed, unjustified complaining about the rules and tools
Quality assurance	Different and unknown practices of partners	Unknown responsibilities for quality assurance, differences in quality of work	Unnecessary duplicate quality assurance due to not trusting others' work

Sharing information	Difficulties in knowing who has the relevant information and who needs it when. Difficulties in defining appropriate means for information sharing	Difficulties in knowing what information should be communicated and how (what is already known and what not). Unclear responsibilities for information sharing	Unnecessary confidentiality, meaning all needed information is not available to all, unwillingness to ask for information if not easily found
Infrastructure	Different incompatible tools used by partners, problems in availability of tools	Different ways of using tools, inability to use communication tools effectively	Blaming tools for problems in the project, unwillingness to use the shared infrastructure
Competence management	Unawareness of what competence is available and where	Suboptimal use of resources, unavailable required competence	Favouring the "known" members of the project although better competence might be available from other sites
Continuous improvement	Different and unknown practices of partners, unclear responsibilities and authorities for improvement work	Unshared information about lessons learned and best practices	Unwillingness to share learning or adopt best practices from other partners

4. GSE solutions

In this section, example solutions for the challenges and problems are presented based on the literature and on the author's empirical work; that is, the industrial cases that were carried out (see details in Section 7). Solutions have been defined for each identified GSE challenge from three sources: (1) adapting from literature, (2) identifying best practices from industrial cases, or (3) by defining new ones if none were available from the first two sources. The first draft of the solutions was presented in the Merlin handbook (Paper III). Updates have been made based on empirical work and new literature sources. The solutions discussed in this section are relevant to all collaboration modes. Supplier-management-related issues are mostly relevant in the customer–supplier relationship, but some of the practices (e.g., co-operation capability, the quality system and roles and responsibilities) are also useful in other collaboration modes such as in multi-site development within a single company and joint ventures.

The solutions are presented from the three viewpoints of technology, processes, and people and for each of the causes presented in section 3. These three viewpoints were chosen as they are commonly accepted as the main viewpoints to consider when improving software engineering or other development practices. The solutions are presented using the same structure as the challenges in Table 4. Table 8, Table 10 and Table 12 together present a comprehensive view of GSE problems/challenges and solutions, i.e., the GSE framework. Figure 4 presents the relationship of the causes and the three viewpoints for solutions.

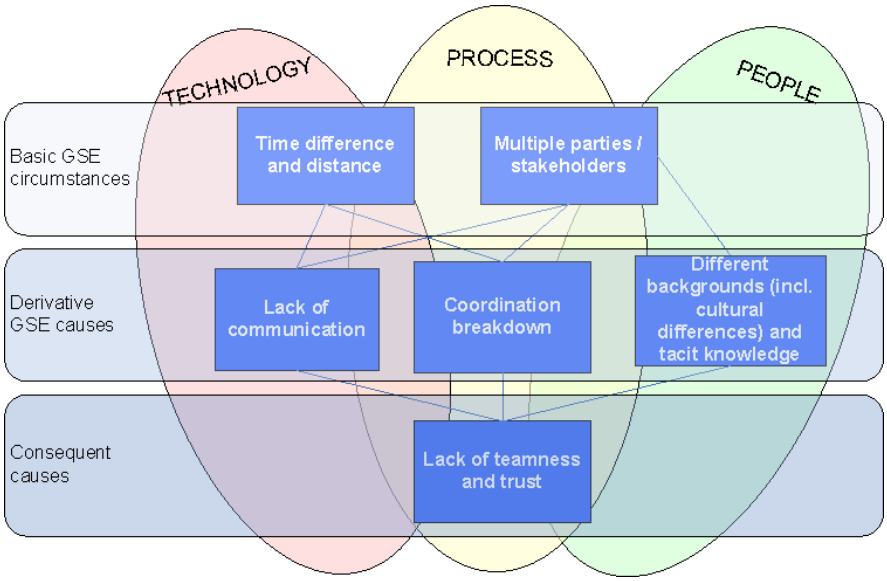


Figure 4. Solution categories' relationship to root causes of challenges.

The figure is based on literature and on the empirical work and it's an indication of the relationship between the types of solutions to the causes of challenges. It does not mean to say that technology solutions are not at all relevant for the challenges caused by different backgrounds and tacit knowledge or that people solutions are not relevant for challenges caused by time difference and distance and lack of communication. The point lies in showing what types of solutions are most relevant to challenges caused by various root causes. The technology solutions can mostly support the challenges caused by time difference and distance and lack of communication, but also, together with process solutions, they can also support the challenges caused by multiple parties/stakeholders and coordination breakdown. People-related solutions can especially support challenges caused by different backgrounds and tacit knowledge, but also other challenges. Relating to lack of communication people related solutions are also relevant, but usually need also technology or process support in GSE (i.e., communication tools, or sharing information practice), Process solutions can support all challenges, but technology and people solutions are needed as well. Furthermore, all types of solutions are needed to address the challenges caused by a lack of teamness and trust.

Next, the solutions are discussed according to the three viewpoints. First, the solutions are discussed in general and then example solutions are presented in a table using the GSE framework structure.

4.1 Process solutions

Process-related solutions can help to address all of the root causes for the problems and challenges in GSE. One solution is usually a partial solution to the problem and should be used in combination with other solutions to completely solve the problem. There is no specific GSE process model available that would address GSE as a whole. Such a model would also not be practical, as it would usually require a complete change in the way of working within a company. Instead, new practices or enhancements of practices are proposed. These practices can be incorporated into any process model, but details of how to address them in practice may vary, for example, Välimäki et al. (2009) propose a pattern based approach to integrate the good practices that support GSE in the company's process and Hossain et al. (2009) discuss how agile practices can help in GSE. The solutions are discussed here according to the GSE framework structure

4.1.1 Management practices in GSE

The GSE view in terms of management practices includes organisational-level activities and project-level activities. These activities are additional activities needed in order to success in GSE. In addition basic activities as presented, e.g., in project management frameworks such as PRINCE2, PMBOK etc. are needed (those are not discussed here). Firstly, a *collaboration strategy* should be defined, including applicable models for collaboration. Collaboration models include, for example, subcontracting, multi-site development, joint ventures etc. (see Section 2.2). A company should have a strategy for which modes to use in which situation, also considering the organisational structure and culture. The organisation should also define a reward policy for collaboration efforts. Collaboration efforts should be rewarded in order to encourage, for example, supporting and helping each other, as these may often be seen as extra work, or work that not enough attention is paid to naturally. Collaboration should also be taken into account in long-term planning and road-mapping: supplier agreements and long-term framework agreements should be used as the input for road-mapping and it could also be useful to consider involving the customers or suppliers in road-mapping, especially in long-term strategic relationships.

Contracting practices are another important topic at the management-practice level, especially in multi-company collaboration efforts. Issues such as ensuring that the contracts are signed at the start of the co-operative venture, involving technical people in the contracting process to ensure that technically realistic agreements are made and agreeing to the change procedures and decision points should be included in the contracting process. During the contracting process all involved parties or stakeholders, with the required authorities, should be involved.

Conditions for collaboration should be established for the organisation as well as for each distributed project. These conditions include communicating the motivation and rationale for collaboration to all parties, establishing a collaboration culture

where collaboration becomes a natural way of working, identifying cultural differences between partners and dividing work across sites or partners in a meaningful way. Basically, all topics discussed in this section are relevant with respect to establishing the conditions for collaboration. Also, defining the confidentiality of the data and allowing or restricting access to it accordingly is important in order to ensure that the required information is available to all those who need it (i.e., not restricting access to all data just in case). This helps in sharing information between partners, when it is clear what can be shared and what cannot be shared.

At management-practice level, practices to *manage the supplier* also need to be defined. Supplier selection should be carried out in a defined and controlled manner, including developing definite criteria for selecting suppliers, analysing suppliers' co-operation capability, and their quality system beforehand and performing supplier audits. Agreements with the supplier should be made in enough detail and technical people should be involved, as they can ensure the technical quality of the agreements. The supplier should also be required to deliver a plan with an acceptable level of detail, enabling the monitoring of the progress of the supplied work. The roles and responsibilities, including the general role of the partners in the project, should be defined clearly and communicated well. The general roles incorporate, for example, mutual responsibilities in partner-to-partner business development vs. a clearly defined implementation or test partner. Roles, responsibilities and authorities on a more detailed level should be defined, including escalation paths in case of issues that are unsolvable at the level under consideration. In addition to these agreements, tracking of the progress of the work during execution is equally important. Paying attention to the supplier, in terms of the planning effort and the time for it, is easily overlooked, so defining explicit tasks and responsibilities for tracking the suppliers' work during the project is needed. This also includes defining acceptance procedures for mutual deliveries and, finally, validation of the supplier's work/results against stated criteria and the evolution of the supplier's performance.

Project management also requires specific attention when doing GSE. The project's organisation should be defined purposefully and functionally for the situation, meaning that the work is distributed between sites so that dependencies between sites are minimised, still utilising the best competencies across sites for various tasks. Project goals need to be explicitly and unambiguously stated and communicated to all teams involved in the project, including the possible partner-specific goals, to enable mutual trust and understanding between partners to develop. Roles, responsibilities and decision authorities should be defined and communicated, including escalation channels in case of issues that are not resolvable at the decision level. Also, critical resources of the sites/partners should be defined and managed. Status reporting and problem-solving practices, including channels and decision authorities should be defined. An important aspect relating to GSE project management is alignment between teams and partners, meaning that the dependencies between teams are identified and managed proactively and that procedures for dealing with deviations are defined. Communication is an important aspect, and communication items, roles and channels (including appropriate tools)

should be defined and potential communication bottlenecks identified and managed. Collaboration-specific risks should be identified and managed as part of the general risk management of the project. Also, shared change-management practices should be defined and communicated to all partners.

As a summary, in Table 5 examples of detailed solutions for how to address the important activities discussed above, in practice, are given. More details of the solutions are presented in the SameRoomSpirit wiki (Prisma, 2011). These kinds of solutions are needed to take care of the important items discussed in this section. The solutions can be specific processes, practices or tools, for example.

Table 5. Sample of detailed solutions to management practices.

Topic	Solution name	Solution description
Collaboration strategy	Characteristics of collaboration modes	Describes main characteristics of selected collaboration modes in order to help in understanding the different modes when choosing them.
Contracting practices	Checklist for subcontracting agreements	List of items to check when making subcontracting agreements; for example, covering IPR issues and acceptance of deliveries.
Conditions for collaboration	Project-initiation practices	Practices to carry out at the start of the project to ensure good conditions to start the project; for example, identification of cultural differences.
Supplier management	Template for supplier reporting	Template describing what suppliers should be reporting in various stages of the project and attention points for detecting potential problems early.
Project management	Project-management milestones	Description of milestones from the GSE viewpoint to track project progress and guidelines for reacting to deviations when reaching the milestones.

4.1.2 Engineering practices in GSE

Specific attention points regarding engineering practices when doing GSE projects are largely related to establishing a common understanding; for example, the requirements and architecture of the product to be made. It is also important that different sites work on the same baseline for the various work items (requirements, architecture, design, code, test material).

Requirements and architecture should be defined clearly, including the non-functional requirements. The relevant experts from all parties should be involved in requirements analysis and prioritisation, and architecture design. Involving persons from each site in the core group defining the requirements and architecture is also useful for ensuring the common understanding and availability of expertise at

all sites. Requirements and architecture should also be continuously communicated and defined clearly and unambiguously in order to ensure a common understanding about them.

Prioritisation rules and practices/trade-off of the requirements as well as practices for identifying and dealing with ambiguous and conflicting requirements should be defined clearly and communicated well. Requirements management practices should also be defined, including change management of the requirements and traceability to enable the tracking of the consistency between requirements and further work products. RE is discussed in more detail as an example area in Section 6.

The architecture should be defined so that it takes into account the collaboration mode, thus enabling the sharing of work between sites. Special attention should be paid to defining the interfaces clearly. Also, the maintenance and evolution of the architecture should be defined and managed.

Relating to *design and coding*, GSE has less impact than for requirements and architecture as the work has less dependencies, and the possible dependencies should have been addressed at the architectural level. However, it is useful to define common design and implementation rules between partners and participate in design and code reviews across sites in order to share information and ensure that the rules are followed.

In *integration* the parts developed by the different teams of the project are brought together and often problems in the common understanding of the requirements and architecture are revealed only then. In order to make integration work easier, the integration strategy should be defined and communicated, the required expertise should be defined and its availability during integration ensured, sufficient time and capacity for integration should be planned and responsibilities for resolving problems should be clearly assigned.

In *testing* it is most important to share information and data between partners; for example, the need for a shared test environment between partners should be identified, and its availability ensured and test cases shared when applicable. Sharing information about the performed tests and the test results is also important in order to avoid duplicate work between sites or grey areas that nobody tests. Relating to the releasing of the product, the costs for non-quality should be taken into account, including the costs for non-quality of the various suppliers.

This discussion is summarised in Table 6, presenting a sample of solutions for engineering practices.

Table 6. Sample of detailed solutions to engineering practices.

Topic	Solution name	Solution description
Requirements engineering	Common understanding of requirements	Describes practices for identifying ambiguous requirements or differences in the understanding of the requirements between partners.
Architecture	Common understanding of architecture	Practices for ensuring a common understanding of the architecture among partners; for example, by utilising various architectural views and design principles.
Design and coding	Common rules	Description of common design and implementation rules that should be followed by the partners in order to ensure a common quality and understandability of work done by others; for example, design patterns to be used.
Integration	Integration strategies	Description of different integration strategies, that helps in choosing the right one for the project.
Testing	Sharing test information	Practices that support sharing test information (test cases, test results, test data) between partners.

4.1.3 Supporting practices in GSE

Supporting practices should be adjusted to support GSE. United *configuration management* (CM) practices between teams and partners are essential for GSE success and they should be defined and trained and their use ensured on a regular basis across sites. Competence in using the CM tool should be available. Also, the release and traceability practices that support the traceability of customer-version-feature/requirement information should be defined and followed across sites.

Common practices for *quality assurance* should also be defined, including, for example, checking the quality of the work on reaching the defined milestones, common reviews and acceptance criteria. However, the common process across sites/partners should be as narrow as possible and forced as little as possible. Also, each partner should have in-house quality practices in place according to the quality requirements of the project.

Sharing information between sites is a specifically important aspect in GSE. One aspect of information sharing is sharing documentation, and practices for that should be in place, including where and when the documents are available or shared and for whom. In order to ease information sharing via documents, common terminology should be used and used abbreviations should be defined. Also, common templates should be defined and used where applicable.

Good *infrastructure* is one of the key aspects for enabling successful GSE. From a process viewpoint, this includes defining the infrastructure needs from

various sites' viewpoints and defining a shared way of working with the infrastructure and tools; for example, where certain information is stored and by whom.

It is also important to ensure that *enough competencies* are available for the project. Thus, the necessary competences should be identified, and potential competence gaps filled with training or through acquiring persons with the right competence. It is especially important to ensure that basic management capabilities are present in teams. Also, the social and communicative skills of project members, especially project leaders, should be considered. For example, ensuring that project members have sufficient language skills is important. In order to ease dealing with the difficulties caused by cultural differences, the differences should be identified and prepared for. Long-term visits between sites help in creating understanding of the other site's culture and way of working. Also, the importance of continuous communication should be emphasised.

As in any development effort, in GSE it is also important to *continuously learn and improve* the practices. Thus, the effectiveness of collaboration should be evaluated, for example, as part of post-mortem project evaluations. The best practices should be recorded and used between partners. Also, in long-term relationships it is useful to agree upon shared process-improvement work.

This discussion is summarised in **Table 7**, presenting a sample of solutions for supporting practices.

Table 7. Sample of detailed solutions to supporting practices.

Topic	Solution name	Solution description
Configuration management	Unified CM practices	Describes the configuration-management practices that should be common between all partners.
Quality assurance	Unified QA practices	Describes the quality management practices that should be common between all partners.
Sharing information	Documentation practices	Documentation practices that make sharing information in distributed projects easier (understandability and availability of documentation).
Infrastructure	Infrastructure checklist	Checklist that helps to ensure that the infrastructure is adequate for the GSE situation in question.
Competence management	Team development	Practices for what to consider when building a team; for example, adequate competences and building a team atmosphere.
Continuous improvement	Evaluating collaboration effectiveness	Metrics and measurement practices that provide information during and after the project about the effectiveness of the collaboration.

4.1.4 Process solutions summary

The following table (Table 8) summarises the challenges and process-related solutions. Table 8 together with tables Table 10 and Table 12 form the GSE framework.

Table 8. Summary of process solutions for GSE challenges.

Management practices	
Collaboration strategy: - Ineffective collaboration due to unsuitable collaboration mode. - Delays and problems in co-operation due to ad hoc way of working.	- Practices for selecting collaboration mode. - Collaboration-mode characterisations. - Defining purposeful project organisation, and collaboration practices fitting to the collaboration mode. - Involving partners in road-mapping to establish trust and co-operative culture.
Contracting practices: - Unclear or non-existent agreements between partners. - Gaps and/or duplicate items in contracts' coverage.	- Practices for establishing good contracts, contract templates. - Practices for agreeing issues not included in the contract, contract change management.
Conditions for collaboration: - Delays and wasted effort due to inappropriate project organisation or missing collaborative practices and tools. - Lack of visibility of partner's work.	- Practices for information sharing, escalation and change management. - Defined responsibilities and authorities, and contact persons for each site. - Defined proactive status-reporting practices and frequent communication.
Supplier management: - Gaps or duplicate work caused by unclear assignments, delays caused by unmanaged dependencies. - No or not enough visibility of suppliers' work. - Unexpected deliveries (content or timing). - Misunderstood requirements.	- Practices for supplier management, including responsibilities, checkpoints, contacts and tasks alignment. - Defined supplier-selection practices also taking into account suppliers' collaboration experience. - Explicit attention placed on managing the suppliers.
Project management: - Unclear status of project. - Delays caused by unclear decision-making authorities and practices. - Delays caused by unaligned teams. - Mismatching work results caused by misunderstood goals. - Unavailable resources.	- Defining a project plan and management structure, including tasks, responsibilities, dependencies and planned effort and schedules. - Practices for aligning teams' and partners' work, practices for managing dependencies between partners, critical resource management.

Engineering practices	
Requirements engineering: - Difficulties in prioritising requirements due to various views. - Unclear requirement documentation. - Difficulties in establishing and managing traceability. - Misunderstood requirements causing unfitting deliveries. - Problems caused by non-communicated changes to requirements. - Inconsistencies between requirements and further work products.	- Requirements prioritisation practices. - Requirements document template covering good requirements quality aspects. - Requirements engineering process for distributed setting. - Practices for identifying and dealing with ambiguous requirements, for resolution of conflicting requirements. - Uniform requirements management practices between teams, including clear requirements change management practices.
Architecture: - Unsuitable architecture (not supporting distributed development). - Unclear, non-communicated or misunderstood architecture.	- Architecture definition practices including checklist for defining architecture for distributed development. - Clearly defined maintenance and evolution of the architecture. - Taking into account the collaboration modes when defining the architecture.
Design and coding: - Difficulties in communicating design rationale causing mistakes in design decisions. - Different coding styles and standards affecting understandability of design and code by others.	- Design and coding guidelines (including design and coding principles). - Review practices to ensure design choices are good. - Shared documentation practices. - Review checklist to ensure standard quality of design and coding.
Integration: - Delays in responses for issues found in integration, unavailability of information. - Problems in integration caused by mismatching pieces or unsynchronised deliveries.	- Defined responsibilities for integration and problem solving. - Defined integration strategy. - Continuous integration practices.
Testing: - Difficulties in replicating defects found in testing. - Lack of visibility in testing progress. - Duplicate testing efforts, inadequate testing (due to lack of knowledge). - Ineffective testing caused by unsynchronised deliveries.	- Defined testing practices for distributed settings, defined responsibilities. - Sharing of test cases. - Practices to share information about the performed tests and the test results. - Clear responsibilities and authorities.
Supporting practices	
Configuration management: - Complex configuration-management practices. - Difficulties in finding correct versions. - Use of wrong versions causing problems in builds. - Unclear status of development work.	- Configuration management process supporting distributed development, including version naming, communication of changes and roles and responsibilities.

Quality assurance: - Different and unknown practices of partners. - Unknown responsibilities for quality assurance. - Differences in quality of work between partners.	- Shared process between partners for certain parts, such as configuration management, change management, requirements management and project management. - Defined interfaces to partners. - Common process for relevant items across sites/partners. - Managing collaboration-related risks.
Sharing information: - Difficulties in knowing who has the relevant information and who needs it when. - Difficulties in defining appropriate means for information sharing. - Difficulties in knowing what information should be communicated and how (what is already known and what is not). - Unclear responsibilities for information sharing.	- Defined information-sharing practices including what, when, from whom, to whom and using what media. - Defined contacts from each partner. - Agreement between partners as to how changes affect contracts. - Practices for assessing impact of change on other parties' work.
Infrastructure: - Different incompatible tools used by partners. - Problems in availability of tools. - Different ways of using tools. - Inability to use communication tools effectively.	- Defined infrastructure needs, defined way of working with the infrastructure. - Defined way of working with tools, regular monitoring of tool-usage practices. - Virtual-meeting practices.
Competence management: - Unawareness of what competence is available and where. - Suboptimal use of resources. - Unavailable required competence.	- Practices to analyse the competence needs as well as for identifying available competences over sites. - Practices for utilising best available competences for the task despite of the competence location.
Continuous improvement: - Different and unknown practices of partners, unclear responsibilities and authorities for improvement work. - Unshared information about lessons learned and best practices.	- Practices for sharing best practices and lessons learned over sites, clear authorities to decide the practices to be used in the project. - Evaluating the effectiveness of collaboration.

4.2 Technology solutions

Technology solutions for GSE are mostly helpful in addressing the challenges caused by time difference and distance and lack of communication. They are also helpful in other challenges, but the main solutions to those are in process- and people-related aspects. Also, technology solutions alone cannot usually solve the challenges, but they need process- or people-related solutions to support them. For example, availability of communication tools does not automatically mean that communication takes place, but their absence hinders communication across sites effectively. Technology solutions are mainly tools and infrastructure issues; for

example, networks and tools that can help in information sharing between partners. The GSE infrastructure should include tools that are available, reliable and usable. Sufficient tools for communication and shared repositories are also very important for successful GSE. The resources should be accessible to all that need them and they should be compatible between partners where needed. Also, the network connections should be usable and reliable. In GSE projects, specific attention should be paid to issues caused by time difference; for example, that appropriate tools for sharing information are available and that the transition between synchronous and asynchronous work is managed.

The tools used in GSE are the same as in single-site development with the addition of communication tools and shared repositories over sites. Important aspects when choosing the tools are the normal requirements for tools, such as usability and fitness for the purpose, but they have to be considered from the GSE viewpoint, meaning that multiple partners with different backgrounds need to be taken into account. Also, partners may have legacy tools, which cannot be changed without huge investments, so interoperability of tools is an important aspect in GSE. Systematic evaluation and selection of tools for GSE project is important, for example, Poston and Sexton (1992) present a workflow for tool evaluation with respect to testing tools that was extended by Winkler et al. (2010) to evaluate pair programming tools for distributed projects. This workflow consists of five steps, analysis and classification of requirements (1), search and categorization of candidate tools (2), evaluation framework definition (3), scenario brainstorming (4), tool evaluation and assessment (5), and can be used in evaluating other tools for GSE as well.

In selecting tools, the following items should be considered (Kanstren et al., 2007):

- Usability, simplicity and customisation: The tool is easy to use and it does not complicate development work. The tool deployment does not require extensive customisation.
- Multi-platform support: The ability to support multiple operating systems.
- Tool integration: Integration with the other tools (SW development, testing, project management) requires that as a minimum the tools should have import/export facilities.
- Web access: The tool has a web interface that makes it unnecessary to install a client application for occasional users.
- Access control: The tool provides access control; thereby each participant has appropriate access to the data (e.g., role-based, project-based or task-based access control).
- Information sharing: The tool supports information sharing and data availability. Thereby all participants can be up to date (e.g., central repository, e-mail notifications, news groups, discussion forums, replication, remote access etc.).

- Simultaneous usage: Many users can access to same system and work on the same data securely.

These criteria are common for all tools; for specific-purpose tools, additional requirements should be considered. For example, for project-management tools the following criteria have been defined (adapted from Ahmad & Laplante, 2006):

- Communication and interaction: The tool supports both asynchronous and synchronous communication. Interaction between project members (negotiation of goals, task allocation, co-work etc.) needs to be supported.
- Detection of critical paths: Support for identifying critical paths is needed.
- Managing of dependencies: In scheduling, the tool supports identifying and managing dependencies.
- Managing of critical resources: The tool supports identifying and managing critical resources.
- Levelling of plans: It is possible to make plans reflecting the project levels (overall project, teams, sites etc.).
- Ability to provide central knowledge repository for file storage, tasks, timelines and resource tracking.
- Ability to create, share, review and redline project documents, check calendars, coordinate schedules and review tasks by project members.
- Permit editing of a document in a parallel manner by project members.
- Support both synchronous and asynchronous group problem solving and decision making.

Ensuring availability of data for all in a globally distributed project can be done by either unifying the tool set between both sites of the project and tools used in different activities by acquiring a bundled tool set from a specific vendor or by building interoperability between various tools used in the project. Several bundled tool sets from various vendors are available, but that often creates a vendor lock and requires that some of the partners change their tools. Changes are usually risky and costly, so creating tool integration is a better potential solution to address GSE challenges from the tool viewpoint.

Interoperability of the development tools is an important aspect even in single-site development as the data in tools is connected and the manual transfer of data between tools is laborious and error prone. In GSE it is even more important. This is due to the fact that multiple sites and/or partners are working together, usually using several tools for the same purpose. For example, sites can be using different CM tools or testing tools due to various reasons. However, the data in these tools needs to be connected to other tools and at least needs to be easily available for different sites.

As discussed in Paper VIII, the interoperability of tools increases the efficiency and transparency of embedded system development; it reduces the amount of laborious and time-consuming manual effort, the number of (manually introduced) errors and reduces the chance of different and incorrect interpretations by providing access to the same data to all parties without needing to replicate it to different tools. Proper tool integration can increase the effectiveness and efficiency of the product development, as it can, for example, improve traceability. Good traceability helps in ensuring that the requirements are explicitly covered by tests, and that the engineering tasks are consistent through the synchronisation and status overviews of successive engineering tasks. Tool integration can also increase the transparency of the project by providing insight into the total project for all involved parties and real-time access of the same data across sites. As a summary, in Table 9 examples of detailed solutions for how to address the important activities discussed above, in practice, are given.

Table 9. General technical solutions.

Topic	Solution name	Solution description
Development tools GSE support	GSE tool requirements	List of requirements that are specific for tools to be used in GSE situation
Data sharing between sites	Tool interoperability	Guidelines and framework for building tool interoperability

The following table (Table 10) summarises the challenges and technology related solutions for each of the GSE framework topics.

Table 10. Summary of technical solutions.

Management practices	
Collaboration strategy: - Ineffective collaboration due to unsuitable collaboration mode. - Delays and problems in co-operation due to ad hoc way of working.	- Discussion forums for questions and answers (that are stored). - Project website introducing project members (with pictures).
Contracting practices: - Unclear or non-existent agreements between partners. - Gaps and/or duplicate items in contracts' coverage.	- None
Conditions for collaboration: - Delays and wasted effort due to inappropriate project organisation or missing collaborative practices and tools. - Lack of visibility of partners' work.	- Establishing working infrastructure, including at least shared configuration management, document sharing and communication tools between partners. - Access to partners' tools for real-time status information.

Supplier management: - Gaps or duplicate work caused by unclear assignments, delays caused by unmanaged dependencies. - No or not enough visibility of suppliers' work. - Unexpected deliveries (content or timing). - Misunderstood requirements.	- Access to suppliers' tools for real-time status information.
Project management: - Unclear status of project. - Delays caused by unclear decision-making authorities and practices. - Delays caused by unaligned teams. - Mismatching work results caused by misunderstood goals. - Unavailable resources.	- Establishing project management tools over sites, such as effort- and task-progress tracking, project-information site. - Shared project management tools, automated indicators for project progress.
Engineering practices	
Requirements engineering: - Difficulties in prioritising requirements due to various views. - Unclear requirement documentation. - Difficulties in establishing and managing traceability. - Misunderstood requirements causing unfitting deliveries. - Problems caused by non-communicated changes to requirements. - Inconsistencies between requirements and further work products.	- Requirements engineering tools that support multi-site working, especially those establishing traceability. - Interoperability of development tools to enable traceability. - Shared requirements repository to ensure availability of correct versions and changes.
Architecture: - Unsuitable architecture (not supporting distributed development). - Unclear, non-communicated or misunderstood architecture.	- Shared repository where correct architecture versions are available. - Discussion forum to share architecture knowledge.
Design and coding: - Difficulties in communicating design rationale causing mistakes in design decisions. - Different coding styles and standards affecting understandability of design and code by others.	- Shared repository where correct design versions and design rationale decisions are available. - Discussion forum to share design rationale.
Integration: - Delays in responses for issues found in integration, unavailability of information. - Problems in integration caused by mismatching pieces or unsynchronised deliveries.	- Shared configuration management tool. - Discussion forum to share integration issues. - Defect management tool to communicate and track defects. - Integration tools and shared automated test sets providing immediate feedback for developers.

Testing: - Difficulties in replicating defects found in testing. - Lack of visibility in testing progress. - Duplicate testing efforts, inadequate testing (due to lack of knowledge). - Ineffective testing caused by unsynchronised deliveries.	- Shared configuration management. system to ensure use of correct version. - Shared test environment.
Supporting practices	
Configuration management: - Complex configuration management practices. - Difficulties in finding correct versions. - Use of wrong versions, causing problems in builds. - Unclear status of development work.	- Configuration management tool that supports distributed development. - Shared websites to share information.
Quality assurance: - Different and unknown practices of partners. - Unknown responsibilities for quality assurance. - Differences in quality of work between partners.	- Shared repositories for QA results.
Sharing information: - Difficulties in knowing who has the relevant information and who needs it when. - Difficulties in defining appropriate means for information sharing. - Difficulties in knowing what information should be communicated and how (what is already known and what is not). - Unclear responsibilities for information sharing.	- Shared repositories and functional communication tools. - Change management tools.
Infrastructure: - Different incompatible tools used by partners. - Problems in availability of tools. - Different ways of using tools. - Inability to use communication tools effectively.	- Interoperability solutions for tools, shared tools for shared work items.
Competence management: - Unawareness of what competence is available and where. - Suboptimal use of resources. - Unavailable required competence.	- Shared repository for competences and their contact information in the project.
Continuous improvement: - Different and unknown practices of partners, unclear responsibilities and authorities for improvement work. - Unshared information about lessons learned and best practices.	- Shared repository of best practices and lessons learned.

4.3 People solutions

People-related solutions are mostly helpful in addressing challenges caused by different backgrounds and tacit knowledge. People issues are important to address in all challenges, as software engineering is highly people intensive work, and thus everything is affected by people aspects. The solutions in this category are mainly related to the competences and training of the people, communication and ensuring people's motivation and attitude for co-operative work, including identifying cultural differences and taking them into account. Solutions relating to these general topics are discussed in Table 11.

As discussed in Paper IV, GSE is very knowledge intensive and there are multiple challenges related to knowledge engineering. The general knowledge engineering technologies can also help in GSE projects. For example, in GSE, the differences in teams'/partners' tacit knowledge should be understood. In order to address the different backgrounds and tacit knowledge, knowledge engineering activities relating to knowledge capture are relevant and to bridge the gaps, knowledge engineering technologies for sharing knowledge are useful. Also, in order to address the motivation of the partners, well-defined roles and activities for reviewing and, thus, sharing the knowledge of the goals of the project are helpful. During the GSE project, a great deal of knowledge is created and needs to be shared between the partners. Thus, solutions that increase communication, trust, openness, and awareness, as well as solutions that make knowledge available in an explicit form are needed. Sharing knowledge can be supported by activities related to negotiating, brainstorming freely and reaching a consensus that can also help in creating a better understanding of each other. In GSE, individual and shared understanding, that is, knowledge creation, requires lots of communication, as communication increases mutual trust between partners. Also, the role of informal communication is important, as it helps to create trust and teamness between partners. Establishing a collaborative culture can be supported by practices for getting to know each other, such as face-to-face meetings (as they make it easier to contact the people in the future), team building practices, rewarding collaboration efforts, and by assigning explicit tasks and responsibilities for communication over sites.

Relating to people competences, identifying the needs for training and arranging them in a timely manner is important. Also, seniors reviewing juniors' work is helpful and the competences should be taken into account in task allocation over sites. Also, critical competences should be identified and managed as part of managing the critical resources.

Table 11. General people-related solutions for GSE.

Topic	Solution name	Solution description
Knowledge	Knowledge engineering	Knowledge engineering technologies to capture and share knowledge at different sites
Competence	Competence building	Description of how to support competence building; for example, via training and peer support
Motivation	Collaborative culture	Description of techniques that help in establishing a collaborative culture

The following table (Table 12) summarises the challenges and technology related solutions for each of the GSE framework topics.

Table 12. Summary of people-related solutions.

Management practices	
Collaboration strategy: - Ineffective collaboration due to unsuitable collaboration mode. - Delays and problems in co-operation due to ad hoc way of working.	- Training people for various collaboration modes. - Communication about collaboration strategy. - Ensuring collaboration competence and attitude.
Contracting practices: - Unclear or non-existent agreements between partners. - Gaps and/or duplicate items in contracts' coverage.	- Training and communicating (creating awareness) about the agreements. - Ensuring contracting competence.
Conditions for collaboration: - Delays and wasted effort due to inappropriate project organisation or missing collaborative practices and tools. - Lack of visibility of partners' work.	- Motivating people for collaboration via, for example, rewards. - Analysing and preparing for cultural differences between partners. - Practices for getting to know each other and for getting used to working as a team.
Supplier management: - Gaps or duplicate work caused by unclear assignments, delays caused by unmanaged dependencies. - No or not enough visibility of suppliers' work. - Unexpected deliveries (content or timing). - Misunderstood requirements.	- Communication about the partners' roles in the project, and about the right level of confidentiality. - Face-to-face meetings. - Supplier representatives' involvement in requirements definition to ensure availability of requirements competence at the supplier's site.
Project management: - Unclear status of project. - Delays caused by unclear decision-making authorities and practices.	- Identifying the required competences and establishing a plan to close possible gaps. - Communicating project goals. - Identifying critical competences and creating backups.

- Delays caused by unaligned teams. - Mismatching work results caused by misunderstood goals. - Unavailable resources.	- Ensuring availability of knowledge of the project goals.
Engineering practices	
Requirements engineering: - Difficulties in prioritising requirements due to various views. - Unclear requirement documentation. - Difficulties in establishing and managing traceability. - Misunderstood requirements causing unfitting deliveries. - Problems caused by non-communicated changes to requirements. - Inconsistencies between requirements and further work products.	- Training on requirements engineering practices. - Ensuring that people with the best competence define the requirements. - Communication of the defined requirements. - Right level and enough involvement of all parties in validation of produced system requirements. - Ensuring competence of requirements and product structure.
Architecture: - Unsuitable architecture (not supporting distributed development). - Unclear, non-communicated or misunderstood architecture.	- Using architectural views to ease understanding of architecture. - Training of GSE effects on architecture. - Development of the architecture by the right people. - Ensuring availability of architecture expertise at all sites.
Design and coding: - Difficulties in communicating design rationale causing mistakes in design decisions. - Different coding styles and standards affecting understandability of design and code by others.	- Availability of expertise at all sites. - Communication and training of the architecture, design rationale and principles. - Availability of the right competences in the reviews.
Integration: - Delays in responses for issues found in integration, unavailability of information. - Problems in integration caused by mismatching pieces or unsynchronised deliveries.	- Ensuring integration competence and availability of developers to solve problems found in integration. - Creating awareness of integration responsibilities. - Availability of architecture expertise at all sites.
Testing: - Difficulties in replicating defects found in testing. - Lack of visibility in testing progress. - Duplicate testing efforts, inadequate testing (due to lack of knowledge). - Ineffective testing caused by unsynchronised deliveries.	- Ensuring availability of testing expertise. - Creating a positive attitude towards testing (not personal evaluation). - Availability of testing expertise and product knowledge at the testing site.

Supporting practices	
Configuration management: - Complex configuration management practices. - Difficulties in finding correct versions. - Use of wrong versions causing problems in builds. - Unclear status of development work.	- Ensuring availability of configuration management competence. - Training on configuration management practices.
Quality assurance: - Different and unknown practices of partners. - Unknown responsibilities for quality assurance. - Differences in quality of work between partners.	- Motivation for quality assurance over sites. - Communication of practices. - Ensuring availability of QA competence.
Sharing information: - Difficulties in knowing who has the relevant information and who needs it when. - Difficulties in defining appropriate means for information sharing. - Difficulties in knowing what information should be communicated and how (what is already known and what is not). - Unclear responsibilities for information sharing.	- Creating awareness of the importance of information sharing and the motivation for doing it. - Social and communicative skills of project leaders.
Infrastructure: - Different incompatible tools used by partners. - Problems in availability of tools. - Different ways of using tools. - Inability to use communication tools effectively.	- Training for the use of the infrastructure. - Availability of competence for solving problems in the infrastructure. - Communication (and other tools) tools training. - Virtual meeting training.
Competence management: - Unawareness of what competence is available and where. - Suboptimal use of resources. - Unavailable required competence.	- Creating awareness of competences over sites and motivation for involving not-known people with the right competences to the tasks.
Continuous improvement: - Different and unknown practices of partners, unclear responsibilities and authorities for improvement work. - Unshared information about lessons learned and best practices.	- Training for the used practices. - Creating motivation to share problems, best practices and lessons learned.

5. Requirements engineering in GSE

In this section RE in general is discussed first and then the challenges in globally distributed RE are discussed as presented in the literature. The RE process is explained in more detail in Paper V. In Section 6, improving RE in GSE is discussed based on the author's empirical work.

5.1 Requirements engineering

RE means activities involved in discovering, analysing, documenting and maintaining a set of requirements for a system (Sommerville & Sawyer, 1997). Requirements management takes care of changes to agreed requirements, relationships between requirements, and dependences between the requirements document and other documents produced during the systems- and software engineering process (Kotonya & Sommerville, 1998). Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its analysis and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration) (Gotel & Finkelstein, 1994).

RE is generally accepted to be the most critical and complex process within the software intensive systems development (Firesmith, 2005; Martin, 1984; Damian, 2002; Standish Group, 2006; Juristo et al., 2002). It is critical, as the quality of the systems is strongly affected by the quality of the requirements, and complex as in RE the most diverse set of product demands from the most diverse set of stakeholders has to be considered. However, in practice, proper RE is not an established practice. As a consequence, many errors are introduced in the requirements' phase, caused by poorly written, ambiguous, unclear or missed requirements. Failure to correctly specify the requirements can lead to major delays, cost overruns, layoffs and even loss of lives.

The basic RE activities include requirements gathering, high-level analysis, allocation and flow-down, detailed requirements analysis, requirements validation and verification and requirements management.

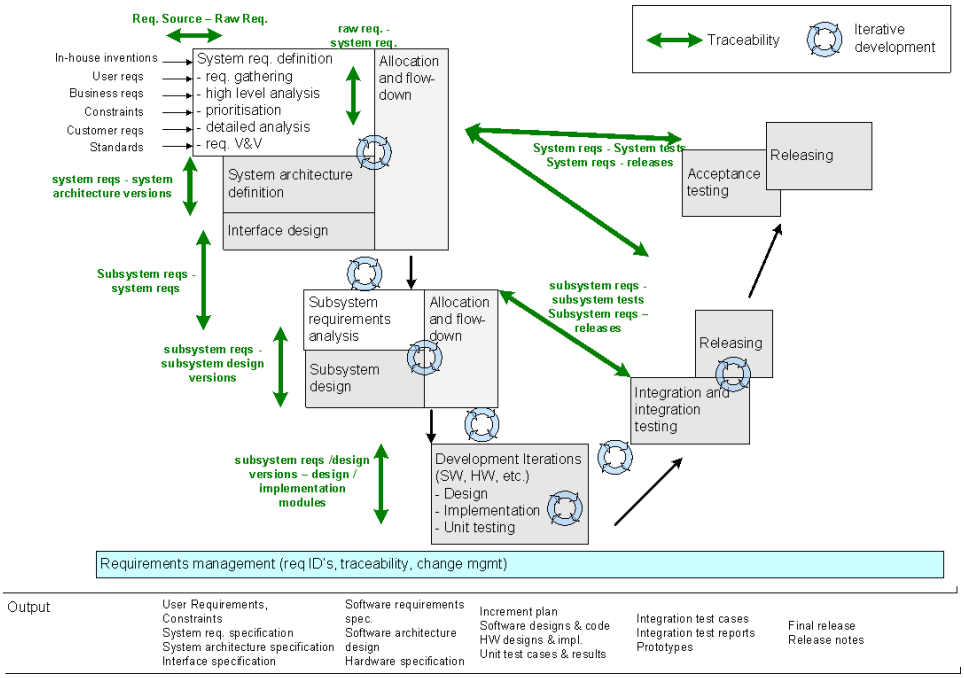


Figure 5. Requirements engineering process.

Requirements gathering (also called elicitation) involves identifying the stakeholders of the system and collecting their requirements for the product (raw requirements). Also, standards and constraints (e.g., the legacy systems) should be addressed. Figure 5 shows the different viewpoints that should be considered while gathering requirements. During the gathering of requirements, some analysis (e.g., cost - benefit and technical-feasibility analysis) is also done.

High-level analysis involves negotiation, agreement, communication and prioritisation of the raw requirements. The analysed requirements need to be documented to enable communication with stakeholders and future maintenance of the requirements and the system. Requirements documentation also includes describing the relations between requirements. During requirements analysis, it gives added value to record the rationale behind the decisions made to ease future change management and decision making.

Allocation and flow-down involves allocating requirements to system components and then defining and validating the detailed subsystem requirements. This activity aims to make sure that all system requirements are fulfilled by a subsystem or by a set of subsystems collaborating together. Top-level system requirements need to be organised hierarchically, helping the parties to view and manage information at different levels of abstraction. Allocation is architectural work carried out in order to design the structure of the system and to issue the top-level system requirements to subsystems. Architectural models provide the context for defining

how applications and subsystems interact with one another to meet the requirements of the system. Flow-down consists of writing requirements for the lower level elements in response to the allocation. When a system requirement is allocated to a subsystem, the subsystem must have at least one requirement (usually more) that responds to the allocation. Allocation and flow-down may be done for several hierarchy levels. The activity starts as a multi-disciplinary activity; that is, subsystems may contain hardware, software, and mechanics but they are first considered as one subsystem. As the allocation and flow-down proceeds down the hierarchy levels, mono-disciplinary subsystems are defined.

Detailed requirements analysis is part of the allocation and flow-down activity, and it enables the specification of the product functions and performance and the establishment of design constraints that the software must meet. Requirements are analysed and documented in more detail in iterations until sufficient detail is reached to enable design.

Requirements validation and verification include validating the system requirements against raw requirements and verifying the correctness of the system requirements documentation. Common techniques for validating requirements are requirements reviews with the stakeholders and prototyping.

Requirements management includes requirement traceability and change management. Requirement traceability means identifying requirements; that is, giving each requirement a unique ID, and then following the life of a requirement in both a forward and backward direction (requirements' source, requirement versions, related requirements, further work products and their versions). Traceability is an important tool for providing information to change management; for example, for analysing the impact of a change proposal, as it helps to define what modules and tests are affected when a change is accepted.

In practice, these activities interleaf with each other and with other product-development areas and are done in iterations. Figure 5 shows RE process activities and their relation to the other product development activities. RE is an iterative process that will go into more detail during each iterative cycle during development phases (iteration is illustrated with the round symbol in the figure). In all phases and iterations, requirements identification and traceability have to be taken into account and the requirements specifications of the different levels should be documented. Traceability is illustrated with the arrow in the figure and the traceability items are listed next to the arrow. The figure also shows the traceability that should be established between requirement levels and other work products. The process is described in more detail in Papers V and VI. There are also various methods supporting these activities and those are discussed in Paper VI.

5.2 Globally distributed requirements engineering

Globally distributed development complicates the RE process, due to the presence of more stakeholders with different backgrounds, for example. Creating a common understanding of the requirements is already a complex task within one company

at one site, but it is even harder when the stakeholders have different tacit knowledge, and time difference makes communication harder. Global software development also further complicates RE due to social and cultural aspects associated with gathering and managing requirements (Hanisch & Corbitt, 2004). López et al. (2009) defined sources for risks of requirements engineering in GSE to be *communication and distance*, that is, risks derived from the dispersion of the stakeholders across countries and time zones*; knowledge management and awareness*, that is, risks derived from the difficulties of keeping awareness, cohesion and coherence of knowledge when different working groups try to access it concurrently; *cultural differences*, that is, risks derived from the interaction among groups where people have cultural backgrounds which vary greatly; *management and project coordination*, that is, risks derived from the establishment of organisational structures, role definition and coordination procedures; *tools which support the processes*, that is, risks derived from the lack of suitable tools which fully support the RE process; and *clients*, that is, risks derived from the interaction with distributed clients.

Several publications have discussed RE challenges in GSE. For example, Cheng and Atlee (2007) give an update to the 2000 RE roadmap paper written by Nuseibeh and Easterbrook (2000) based on extensive state-of-the-art study. In their paper, they have identified globalisation as one of the hot research topics for RE. According to them, globalisation poses two main challenges to the RE research community. The first challenge is that new or extended RE techniques are needed to support the outsourcing of downstream development tasks such as design, coding and testing. Distance complicates the development, particularly if the teams are from different organisations, have different cultures or have different work environments. Poorly defined requirements are likely to be misinterpreted, resulting in a system that does not meet the stakeholders' needs. The second challenge is to enable effective distributed RE. Requirements activities are, and will be globally distributed, since requirements analysts are usually working with geographically distributed stakeholders and distributed development teams may work with in-house customers. As such, practitioners need techniques to facilitate and manage distributed RE, not just geographically distributed but distributed in terms of time zone, culture and language. However, there are not many studies of the RE methods and tools from the GSE perspective; that is, how well they support GSE.

Bhat et al. (2006) state that RE teams working in client–vendor outsourcing relationships face challenges that traditional software engineering practices do not directly address. The first common characteristic is that a vendor typically faces two stakeholders from the client organisation: the client's IT group and its business community (managers and users). The second common characteristic is the existence of multiple RE processes and tools across organisations and locations. This impacts RE in two ways. Having multiple tools, templates and methodologies that do not integrate or interoperate can lead to wasteful RE rework or loss of data during transfer from one tool to another, which can increase requirements' defects. The third common characteristic is the differences in organisational culture or

location-specific work cultures between the client and vendor teams. The fourth common characteristic is ad hoc staffing of client and vendor team formations leading to multiple transitions across locations between the outgoing and incoming team members.

Yousuf et al. (2008) have studied the existing requirements validation techniques. Their results show that some of the existing requirement validation techniques require extensive informal communication that is not convenient in GSE, while other techniques which can be applied effectively in GSE are not yet very mature.

Damian and Zowghi (2003) report findings of a case study of two multi-site development organisations where groups of customer, product-management and engineering-specified requirements come from remote locations. They describe the challenges faced by the stakeholders in activities such as requirements elicitation, analysis, negotiation and specification. These challenges are depicted in Figure 6 with their relationship to RE activities and known problems in GSE.

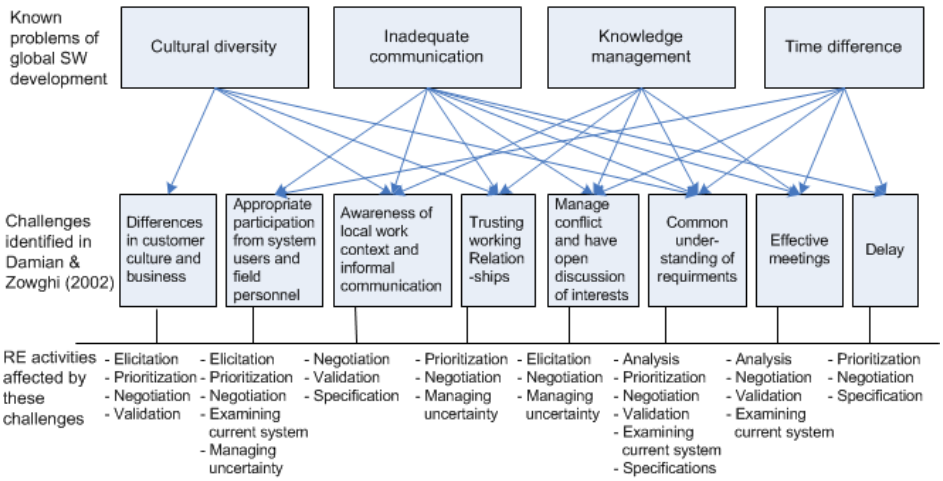


Figure 6. Challenges and impacted RE activities (Damian & Zowghi, 2003).

Next, these known problems are discussed in more detail, including similar challenges reported by others.

Cultural diversity: Differences in stakeholders' language and national culture affect global collaboration. An additional important aspect is the impact of differences on the organisational and functional culture. These factors contribute to a fundamental problem in RE: requirements being expressed using diverse terminologies and levels of detail, thus making the analysis for consistencies, conflicts and redundancies difficult (Damian & Zowghi, 2002). Learning to combine RE techniques across national, cultural and language borders presents unique challenges for developers. Cultural differences have been studied by many researchers with similar conclusions to Brockmann and Thaumüller (2009), who report that

team members with similar cultural expectations, such as those from different western European countries, had less difficulty working together. The larger the cultural distance between team members, such as those between China and Germany, the more difficulty they had in coordinating their different styles of communication, conflict-management strategies and especially in how individual members dealt with criticism and ambiguity.

Inadequate communication: RE requires a higher degree of communication than the other systems development activities, making it more complex in global teams. Problems occur with requirements changes in global software development because “it is hard for the formal mechanisms of communication, such as specification documents, to react quickly enough” (Mockus & Herbsleb, 2001; Hanisch & Corbitt, 2004). Distance complicates informal and face-to-face communication, as the stakeholders’ communication is often dependent on the quality of using synchronous or asynchronous electronic communication tools. Furthermore, the interplay between culture and distance impacts on the ability to reconcile different viewpoints with regards to requirements as well as requirements processes. The resolution of conflicts not only requires good communication during the project but also an understanding of these cultural differences and how they can be overcome (Damian & Zowghi, 2003). Hanisch and Corbitt (2004) describe experiences of large distributed projects. According to their study, the main impediment to RE during global software engineering is communication. Communication issues may be further described in terms of four categories: distribution of the clients and the development team, distribution of the development team, cultural differences between the clients and the development team and cultural differences among the development team.

Knowledge management: Sharing the large amount of information and knowledge about requirements from multiple sources at remote customer sites is difficult to share appropriately with the developers. Also, if experienced developers are not available at all sites, miscommunication and misinterpretation of requirements are likely to occur at the location where team members are less experienced. Availability of the key users may also be an issue in distributed development; if they are not available, the requirements may be easily biased towards the users who are available (Damian & Zowghi, 2003).

Time difference: Large time-zone differences allow little overlap for synchronous collaboration. Hence, asynchronous channels have to be predominant in the communication, complemented by regular teleconferencing calls. Thus, when time difference is high, there is a tendency to rely on written documentation, which has been found to be a very poor way in which to communicate requirements clearly (Damian & Zowghi, 2003). Time difference may also cause high communication overheads (Hanisch & Corbitt, 2004).

6. Improving global requirements engineering

The challenges in globally distributed RE were discussed at a general level in Section 5.2. In this section, the RE challenges are first discussed and elaborated based on literature presented in Section 5.2 and the empirical work carried out by the author. Second, relevant solutions for RE in GSE are discussed. This section describes RE viewpoint of GSE framework in more detailed level, the challenges of requirements engineering activities are presented from the GSE root causes viewpoint, in order to identify solutions for RE in GSE.

6.1 Challenges

In this section, the impact of basic and derivative circumstances and consequent causes are discussed according to the main RE activities (requirements gathering, high-level analysis, requirements allocation and flow-down (detailed analysis), requirements validation and verification and requirements management [identification, traceability, change management]).

6.1.1 Basic GSE circumstances

The basic GSE circumstance of *time difference and distance* affects **requirements gathering** by limiting the possibilities for face-to-face discussions about the requirements, and thus putting higher demands on the documentation of the requests. On the other hand, if requirements gathering is distributed, it offers the possibility of gathering requirements from a wide set of stakeholders from various places through face-to-face discussions. However, the knowledge of the requests is then also distributed and needs to be shared. *Multiple partners/stakeholders* affect requirements gathering so that on the one hand, more time is required to address their needs, and on the other hand it enables a wider view of the needs regarding the product to be built. Also, identifying the stakeholders and their priorities (whose requirements should be most important) is more challenging when there are many stakeholders to consider and multiple partners giving their opinion on the matter.

High-level analysis is more complicated due to *time difference and distance,* because the availability of more information concerning the raw requirements that is gathered may not be available on site, but in some other location. This makes the requirements' analysis more vulnerable to interpretations and assumptions that may not always be correct, thus leading to the wrong requirements or the wrong emphasis on the requirements. Also, there is a challenge that the requirements from stakeholders who are closer are unjustifiably prioritised as being more valued than the others. *Multiple partners/stakeholders* affect high-level analysis of the requirements by giving multiple views on the requests. This can be both positive and negative: positive in the respect that more views are involved in the analysis, making the analysis more comprehensive, but negative because the views and needs of the stakeholders and partners may be contradictory, requiring more time and compromises to reach an agreement on the requirements. Also, communicating and establishing a common understanding of the requirements among the partners requires more efforts than in single-site development.

In **allocation and flow-down (detailed analysis)***, time difference and distance* and *multiple partners/stakeholders* cause similar challenges as in high-level analysis. However, the availability of further information from stakeholders may be of less importance if the high-level analysis is done properly. Also, in allocation and flow-down the distribution should be taken into account to avoid unnecessary dependencies between sites, for example. Additionally, if the analysis work is distributed over sites, *time difference and distance* affects the ability to communicate with other analysts. *Multiple partners/stakeholders* mean that the analysis results may not be the same, and may not be understandable to the others and also that the relevant knowledge may not be available to all partners.

Requirements validation and verification (V&V) are affected by *time difference and distance* so that involving all of the relevant stakeholders from different sites in the V&V work is difficult. Thus, again, the role of documentation is important. Also, sharing work between *multiple partners/stakeholders* needs attention in order to utilise the best competences and resources optimally; that is, to avoid duplicate work.

Regarding **requirements management**, *time difference and distance* and *multiple partners/stakeholders,* challenges are created regarding traceability, as developing the traceability requires knowledge about the product structure and the development process artefacts, which may need to be shared over sites. Good management of traceability is important, so that the work is done based on the correct information, when the background understanding of the people involved is not necessarily the same. Creating and maintaining traceability is also easily neglected, and thus communication about the importance of proper traceability is essential. Also, various tools used by the different partners/stakeholders may complicate the situation. Analysing the impact of changes becomes more complicated when more sites in different time zones are involved; for example, due to the availability of the required competences. Also, implementing and communicating about the changes requires more effort in distributed settings.

6.1.2 Derivative GSE causes

The derivative GSE cause of a *lack of communication* affects **requirements gathering** by adding problems to the work-sharing and documentation aspects of the raw requirements. *Coordination breakdown* can cause duplicate work or gaps in requirements gathering. *Different backgrounds and tacit knowledge* can cause problems in the identification of requirements (e.g., what is a requirement and what is not, and assumptions on what is truly a part of the product based on previous versions of the product) and description of the gathered requirements.

Lack of communication during **high-level analysis** can cause misinterpretations of the requirements, wrong prioritisations of the requirements and ambiguous or conflicting requirement descriptions. *Coordination breakdown* can cause suboptimal use of resources in the analysis (e.g., distributing analysis work suboptimally with respect to the availability of stakeholders), duplicate work or gaps in covering the requirements. *Different backgrounds and tacit knowledge* can cause misinterpretations of the requirements and their priorities, wrong assumptions, as well as requirements descriptions that are not of high enough quality or that contain too many (technical) details. Also, effort may be wasted if the work is not properly coordinated. The effect of these circumstances in **allocation and flow-down (detailed analysis)** is the same as in high-level analysis.

In **requirements validation and verification**, a *lack of communication* can have an affect so that V&V results are not shared, meaning that the corrective actions are not taken, or that V&V work is done on the wrong version of the requirements. *Coordination breakdown* can affect V&V so that resources are not used optimally; for example, V&V work is waiting for some other task to be completed or the best competence for V&V is not available in a timely fashion. *Different backgrounds and tacit knowledge* can affect the quality of the V&V work as well as the interpretations of the requirements and thus can lead to wrong conclusions being made based on V&V results.

Requirements management is affected by a *lack of communication* in many ways; for example, requirements traceability may not be implemented, changes to requirements may come as a surprise later on, thus leading to incorrect implementation, non-communicated assumptions or interpretations of requirements made may lead to incorrect implementation and so on. *Coordination breakdown* has a similar impact and *different backgrounds and tacit knowledge* affect change-impact analysis, where the wrong assumptions of non-impact can be made and can affect traceability, where the wrong or inadequate traceability links may be created.

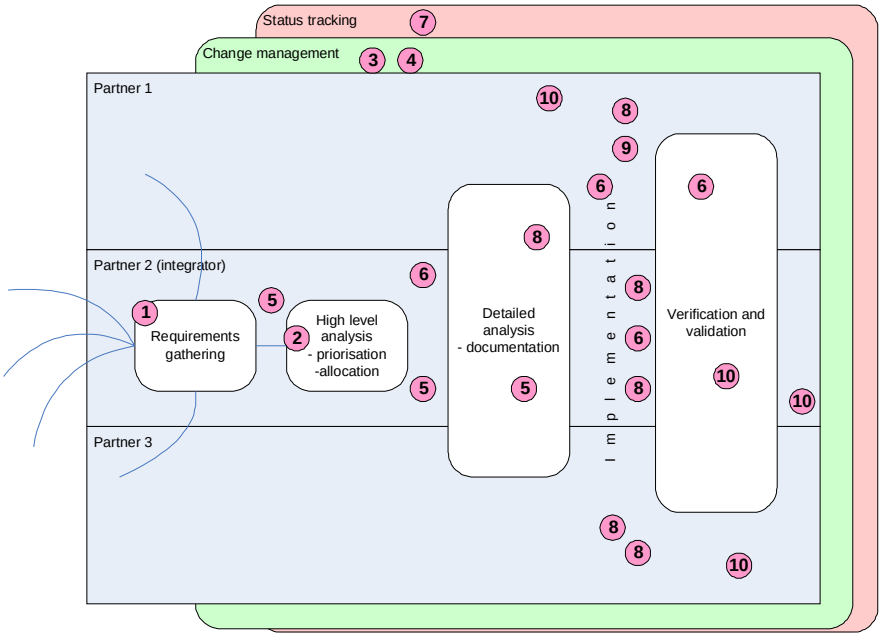
6.1.3 Consequent cause

The consequent cause of *lack of teamness and trust* can affect all RE activities severely, as it hinders the openness to share both the knowledge of the product development, including the stakeholders' needs, and of problems encountered during the project. For example, if there is no trust between teams, the teams are

neither willing to help each other, nor willing to ask for help or show their uncertainty over something. This affects RE so that teams would rather make their own – possibly wrong – assumptions on ambiguous requirements rather than ask for clarifications. Also, in the case of problems that are found in requirements V&V, the teams would rather blame each other than constructively try to find the cause of the problems and learn from them.

6.1.4 Example situation

A simplified picture of RE activities is presented in Figure 7 along with challenges that may appear during the activities. As can be seen from the figure, there are more challenges as more interaction with various partners is needed.



1. Gathering requirements from all relevant stakeholders
2. Prioritising requirements correctly
3. Communication of changes
4. Change impact analysis
5. Clear description of requirements
6. Dividing work (based on requirements) wisely
7. Visibility to partners work
8. Ensuring correct understanding of requirements
9. Availability and use of correct versions
10. Availability of information

Figure 7. Example requirements engineering challenges.

These challenges were discussed earlier in this section, and as can be seen, some of the challenges appear in multiple places during RE activities. For example, a clear description of requirements (5) and availability of information (10) affect the

entire process. Both of these challenges are basically caused by multiple partners, who have generated an increased need for communication and coordination of the work. If specific attention is not paid to communication and coordination, more challenges appear.

6.2 Solutions

In this section, solutions for the challenges caused by GSE to RE are discussed from process, people and technology viewpoints. The solutions are based on literature and empirical work (industrial case) as the solutions discussed in section 4.

6.2.1 Process related solutions

Requirements gathering and high-level analysis: Distributed development limits possibilities for face-to-face discussions about the requirements, thus placing higher demands on the documentation of the requests. As discussed in Paper VII, several templates and criteria for good requirements documentation exist and can also be utilised in globally distributed RE without specific tailoring for GSE. Templates that help to address relevant topics when gathering stakeholders' needs are helpful because they ensure standard practice in collecting the needs and thus the work can be shared between partners. Techniques that help in identifying the stakeholders and their priorities (whose requirements should be the most important) have also been created (examples, such as QFD (Revelle et al., 1998), SCRAM (Sutcliffe, 1998) and Volere (Robertson & Robertson, 1999), are discussed in Paper VI). These techniques also then help to prioritise requirements based on calculated weights including stakeholder priorities, thus giving an objective view of the priorities, instead of a subjective estimate of a person. They also help in taking into account the multiple viewpoints that need to be addressed in prioritisation. Also, communicating priorities to other sites is important, so that the work is done based on the correct priorities.

Allocation and flow-down (detailed analysis) and requirements V&V: Ensuring availability of information to all partners can be supported by identifying information needs and information owners, as well as communication responsibilities and means. Good requirement descriptions are very important in GSE, as they are an important means of sharing information. Thus, proper requirements documentation practices, including defining properties of a good requirement description and the relevant requirement attributes are helpful. These aspects are discussed in Paper VII. Describing non-functional requirements well is even more important in GSE, so that they will be taken into account in everyone's work. Non-functional requirements are vulnerable to different kinds of interpretations, which are more likely in GSE due to the different backgrounds of the people involved. Avoiding and identifying ambiguities, conflicts, wrong interpretations and assumptions and ensuring a common understanding of the requirements can be supported by common terminology, document templates and reviewing each other's work over

sites. Also, requirements V&V technologies, such as prototyping, simulation and reviews are helpful.

Requirements management: Analysing the impact of changes can be supported by proper traceability. Establishing full traceability is, however, laborious. Heindl & Biffl (2006) address requirements tracing options and propose concepts for enhanced requirements tracing that include the rationale for requirements, related decisions, their history, and stakeholder value propositions. They also present a cost-benefit model that helps the project manager to understand what tracing approach is worthwhile to address requirements risk in a project. Use of checklists and defined procedures can also help in discovering any possible implications of a change. When multiple teams or partners are involved, a levelling of the change-requests' analysis is important in order to optimise the use of resources and to ensure an adequate level of communication, meaning that changes are managed at their level of impact. For example, changes that affect only a subsystem can be managed at that subsystem level, but changes that are implemented only in a subsystem, but affect its interface to another system, need to be managed at a level where the affected subsystems are also represented. Thus, a clear definition of what change or problem is to be handled at what level, including criteria to transfer these to other levels, should be made. Also, the presence of enough competence when making these decisions should be ensured.

6.2.2 Technology related solutions

In order to address all the stakeholders' needs, many tools that support requirements gathering are available. Also, many requirements management tools exist. In order to support RE in GSE, an RE tool should include features such as a general repository, import/export facilities, communication capabilities, access-control mechanisms, change-control mechanisms and information sharing mechanisms. Compatibility of the various tools used by the different partners/stakeholders should be ensured. Tool interoperability is discussed in Paper VIII.

Avoiding the use of the wrong versions of requirements or background information can be supported by shared repositories and interoperable tools that ensure that the correct versions are available for everybody. However, communication of the changes and correct document versions and their locations are still needed.

Change management tools can help in sharing information as well as with the implementation work among partners. They can also help in monitoring the progress of implementation, when information from the tools can be shown to all in real time.

6.2.3 People related solutions

In GSE, the knowledge of the requests is distributed and needs to be shared and there are often multiple views on the requests. Sharing this knowledge can be supported by frequent communication and discussion of the requirements. Also,

involving people from each site in requirements analysis helps to share information and ensures the availability of on-site expertise about the requirements. Likewise, analysing the impact of changes can be supported by involving the relevant experts in the change analysis.

In order to utilise the best competences and resources optimally, they should be identified and managed as part of the project management. People from different cultures and backgrounds do not necessarily understand things in the same way, so it is essential that requirement descriptions are unambiguous, consistent and clear. This can be supported by requirements analysis and documentation training. Also, ensuring a common understanding is important, although challenging, as partners may not be willing to communicate on unclear issues, or they may not be aware of different interpretations of the requirements until late in the project.

Validation of each individual project stakeholder's interpretation of the requirements before implementation takes place can be supported by frequent communication and by ensuring the availability of relevant expertise. This is all very knowledge intensive; it requires proper knowledge creation and, specifically, the correct transfer of knowledge in the distributed development situation. This is discussed in more detail in Paper IV.

6.3 Summary of RE challenges and solutions

As a summary, in Table 13 examples of relevant solutions for how to address the important activities discussed above, in practice, are given. Detailed practices are described in the SameRoomSpirit wiki (Prisma, 2011).

Table 13. Summary of RE challenges and solutions.

Challenges Time difference and distance (B1), multiple parties/stakeholders (B2), lack of communication (D1), coordination breakdown (D2), different backgrounds (D3), lack of teamness and trust (C1)	**Solutions** Process (PR), people (PE), technology (T)
Requirements gathering - Limited possibilities for face-to-face discussions (B1) - More time required to address all stakeholders (B2 - Plentiful set of requirements (B2) - Problems in work sharing and documentation of the raw requirements (D1) - Duplicate work or gaps (D2) - Problems in identification of requirements (D3)	- Techniques to identify all relevant stakeholders and set their priorities (PR) - Communication tools and guidelines (T, PR) - Identifying information needs and owners and defining communication responsibilities and means (PR) - Tools for distributed requirements' gathering and shared repositories (T) - Frequent communication (PE)
High-level analysis - Availability of information (B1) - Interpretations and misunderstandings (B1) - Multiple views on requirements and their priorities (B2) - More effort required to establish a common understanding (B2) - Misinterpretations of the requirements (D1, D3) - Wrong prioritisation of the requirements (D1, D3) - Ambiguous or conflicting requirement descriptions (D1, D3) - Suboptimal use of resources (D2) - Duplicate work or gaps (D2) - Incorrect assumptions about requirements (C1)	- Common terminology and templates for requirements documentation (PR) - Distributed review practices (PR) - Information management practices (PR) - Prioritisation techniques and tools (PR, T) - Involving people from each site (PR) - Requirements for RE tools in distributed environment (T) - Shared repositories (T) - Frequent communication (PE) - Training (PE)
Allocation and flow-down (incl. detailed analysis) - Less possibility to communicate with other analysts (B1, B2) - Differences in analysis results (B2) - More effort required to establish common understanding (B2) - Misinterpretations of the requirements (D1, D3) - Wrong prioritisation of the requirements (D1, D3) - Ambiguous or conflicting requirement descriptions (D1, D3) - Suboptimal use of resources (D2) - Duplicate work or gaps (D2) - Incorrect assumptions about requirements (C1)	- Common terminology and templates for requirements documentation (PR) - Information management practices (PR) - Distributed review practices (PR) - Involving people from each site (PR) - Shared repositories (T) - Frequent communication (PE) - Establishing a collaborative culture (PE) - Training (PE)

Requirements V&V - Involving all relevant stakeholders from different sites (B1) - Challenging to utilise the best competences and resources optimally (B2) - Non-implemented corrective actions (D1) - Use of wrong versions (D1) - Delays caused by needing to wait for other parties' input (D2) - Suboptimal use of resources (D2) - Duplicate work or gaps (D2) - Poor quality of V&V work (D3) - Poor quality of releases (D3) - Problems in finding defect causes (C1)	- Managed dependencies between parties (PR)\ - Requirements validation and verification technologies such as prototyping, simulation and reviews (PR) - Information management practices (PR) - Shared repositories (T) - Interoperability of tools (T) - Frequent communication (PE) - Competence management practices (PE) - Establishing a collaborative culture (PE) - Training (PE)
Requirements management - Availability of product knowledge (to make the right traceability links) (B1) - Establishing and maintaining traceability over partners' borders (B2, D1, D2, D3) - Analysing impact of changes and informing about the changes (B1, B2, D1, D2, D3)	- Checklists for requirements change-impact analysis (PR) - Practices for levelling of change requests (PR) - Information management practices (PR) - Shared repositories (T) - Requirements for RE tools in distributed environment (T) - Compatibility of RE and development tools (T) - Shared change management tools (T) - Frequent communication (PE) - Establishing a collaborative culture (PE) - Training (PE)

7. Empirical results

The empirical results are from two phases: in first phase an industrial inventory, including industrial experience reported by others in the literature was carried out. Based on this, an initial structure for the GSE framework was developed, including the main challenges to be addressed in GSE projects and solutions for them.

Next, industrial cases were carried out for specific GSE challenges. These cases are introduced in Table 14. The experiences and solutions tried out in the cases were then incorporated into the framework, creating the first version of the GSE framework (Merlin collaboration handbook), which is discussed in Paper III. Then another set of industrial cases were defined in the Prisma project with different companies, sometimes addressing the same topics as in the first set, thus validating the first version of the framework but also addressing different challenges and solutions, thus adding to the framework. These experiences were then incorporated into the framework, creating the second version of the GSE framework. Altogether, 52 industrial cases relating to distributed development were carried out during the projects over the years 2004–2011. The cases are discussed in detail in section 7.2.

7.1 Industrial inventory

The industrial inventory addressed GSE practices and challenges in seven companies via a questionnaire and interviews. A total of five companies participated in the inventory and 12 interviews with senior managers, project managers, software developers and testers were carried out. In addition to the interviews, two companies filled in a questionnaire, which was also filled in for the five interviewed companies. The industrial partners represent several divergent embedded SW business areas: embedded data-management solutions, telecommunications, embedded SW subcontracting, and consumer electronics. In addition to this, an extensive study of the literature concerning GSE experiences was carried out. Based on this literature study, practices and challenges from 15 cases reported by others were included. The findings of this inventory were the basis of the industrial cases and also provided a first set of challenges and solutions for GSE documented in the first version of the GSE framework. The inventory was based on a framework

defined in the GSE literature using CMMI as the basic structure. The items covered included management practices (e.g., contracts, project management, collaboration management), engineering practices (e.g., RE, architecture, integration, testing) and supporting practices (e.g., configuration management, change management, infrastructure, co-operative work). Also, general information about the GSE environment was addressed such as collaboration modes used and the motivation for collaboration. The most common collaboration mode was product structure-based subcontracting; that is, work was subcontracted to a 3rd party based on requirements or on the implementation of a product part. The most common motivations for collaboration were to reduce costs, to acquire competence (technology or knowledge of a certain market) and to avoid investing in a company's non-core competence areas. The main risks in GSE were seen to be a lack of trust and openness of communication between partners, unclear assignments or specifications of work in contracts, the collaboration lasting in the future, the quality of the acquired product, for example, reliability and performance, becoming too dependent on one partner, for example, getting enough priority and that one's own competence weakens when development has been outsourced. Also, critical factors for collaboration to be successful were identified, including the mutual benefit from collaboration and partners complementing each other's expertise.

The most important/complex points in GSE were seen to be taking into account and prioritising the customers' future requests or suppliers' future directions, defining as detailed a specification of the work as possible (with reasonable effort), the good management of dependencies, status-reporting practices and change-management procedures, for example, channels and escalation, developing and implementing an efficient integration strategy taking into account the added complexity (e.g., new actors) and change management, due to, for example, the scope of impact assessment, communication and relevant viewpoints in decision making.

Relating to solutions, the most commonly defined collaboration practices in the inventoried companies were change management, progress reporting, customer-support processes (suppliers) and, sometimes, meeting practices and participating roles and common version management practices. Usually there were no specific tools for collaborative development and e-mail and phone were the tools for communication. However, the companies used the same tools between partners for defect management, configuration management, requirements management and change management and it was said that most tools did not support collaborative development well. The most important items or areas that were seen as needing most development work or a further search for solutions were explicit statements of project goals, clear task assignments and responsibilities, managing critical resources, defining and ensuring the right level of confidentiality, defining clear and fixed requirements, clear prioritisation rules and practices for the requirements in the case of many interest groups, the effective sharing of a test environment, results and resources and choosing the right integration and testing strategy.

7.2 Industrial cases

7.2.1 First set of industrial cases

In the first phase (see Table 14), 36 cases were carried out to evaluate the solutions identified for the challenges revealed in the industrial inventory. For each case, the topic, GSE challenges addressed, solutions tried out, company domain and author's role in the case are presented.

Table 14. Overview of the first set of industrial cases.

Topic and GSE viewpoint	***Solutions***	***Company domain and author role***
Cases 1, 2, 3 and 4 Improving the distributed requirements engineering process from various viewpoints; for example, flexible and easy to tailor to specific development approaches/environments of different customers, and a specific focus on performance requirements. Quality of requirements descriptions is essential in GSE as it is often the main input for the work carried out at different sites.	Requirements engineering practices for distributed development.	Four cases in four large companies from telecommunications, IT services and real-time embedded-system domains. Author has been actively involved in three of the cases (action research); one case is included based on an experience report.
Cases 5, 6 and 7 Define solutions for early RE phases (gathering, prioritisation and recording methods, and gaining a better understanding of customers' and other stakeholders' needs). The challenge is that the multiple stakeholders' viewpoints should be taken into account with the correct weighting and understood correctly.	Prioritisation methods and tools.	Three cases in two SMEs in data-management solutions and sensor-development domains. Author has been actively involved in two of the cases (action research) and one case is included based on a case report.
Cases 8, 9, 10 and 11 Improving requirements management, including better traceability, especially in partnering and subcontracting business situations. In a multi-site project it is more difficult to establish traceability over sites and the quality and timing (right time input) of the requirements description is essential.	Requirements management practices for distributed project, experiment on automating requirement management.	Four cases in three large companies in semi-conductor, consumer electronics and telecommunications domains. Author has been actively involved in two of the cases (action research); two cases are included based on experience reports.
Cases 12, 13 and 14 Improving the handling and evaluation of non-functional requirements in collaborative projects.	Non-functional requirements identification and description techniques.	Three cases in two SMEs from mobile and wireless systems and data management solutions domains.

Good description of non-functional requirements is essential in GSE, as otherwise interpretations and assumptions will be made.		Author has been actively involved in one of the cases (action research); two cases are included based on experience reports.
Cases 15, 16, 17 and 18 Improving architecture definition, including improving the description and validation of requirements for the architecture. Improving architecture adaptability. Architecture should support multi-site development and be understandable to all partners.	Utilising architecture adaptability mechanisms. Architecture evaluation framework to ensure architecture quality.	Four cases in three large and one SME company from IT services, real-time embedded systems, telecommunications and mobile and wireless systems domains. These cases are included based on the case reports.
Cases 19, 20, 21 and 22 Improving requirements documentation and design documentation to contain the essential information. In multi-site projects, documentation is an important means of information sharing, and thus should be usable.	Design method and tool that help in systematic design description	Four cases in three large and one SME companies from consumer electronics, semiconductors, telecommunications and electricity network equipment domains. Author has been actively involved in one of the cases (action research); three cases are included based on experience reports.
Case 23 To define and test methods, techniques and tools for product and process measurements with an emphasis on automating the measurements. Measurements are especially important in GSE, as they can provide accurate and real-time information from the various sites.	Metrics and tools as well as measurement and analysis process adapted to GSE environment.	One case in an SME from the data management domain. This case is included based on the case experience report.
Cases 24, 25, 26, 27 and 28 To improve the quality, controllability, efficiency and affordability of collaboration with partners and to enhance the ability of distributed teams to collaborate more effectively. Improve description of statement of work. Teams' ways of working in GSE in general. Statement of work is often the basis for work of subcontractors and thus it is very important that it is defined well.	A framework to assess the subcontractor from quality and process perspectives. Merlin collaboration handbook to find solutions for the situation in this case. Statement of work checklist.	Five cases in one large and one SME company from telecommunications and data management domains. Author has been actively involved in three of the cases (action research); two cases are included based on experience reports.
Cases 29, 30 and 31 Improving tool support from testing viewpoint in distributed projects. A lot of effort is wasted in GSE on testing, when different/wrong versions are used in	Automated build and test suite. Tool integration platforms and solutions.	Three cases in two large and one SME company from consumer electronics and data management domains.

replicating defects or when similar tests are run due to not knowing what tests others have performed.		These cases are included based on the case reports.
Case 32 To improve defect management practices and tool support in distributed projects. Defect analysis is more difficult in distributed development; also communication of the defects with partners is more challenging.	Defect management practices for GSE, tool requirements for GSE-supporting tools.	One case in an SME from mobile and wireless systems domain. This case is included based on the case experience report.
Case 33 Improve managing IPRs in collaborative software development. Proper IPR management is important in order to ensure proper confidentiality and thus trust.	IPR management practices	One case in large company in telecommunications domain. This case is included based on the case experience report.
Case 34 Improve configuration management in distributed collaborative projects. Shared configuration management between partners/sites is very important in GSE as it helps to ensure consistency of work, to detect problems early and to track progress of work.	Configuration management practices in GSE, configuration management tool requirements from GSE viewpoint.	One case in large company in consumer-electronics domain. Author has been actively involved in the case (action research).
Case 35 Improve the release planning process through analysis of past experiences. In GSE it is also important to take into account learning and best practices from all partners.	Structured technologies to analyse learning taking into account the GSE situation.	One case in large company in telecommunications domain This case is included based on the case experience report.
Case 36 Improving tool support for collaborative projects. Tools are important for information sharing in GSE and integrated tools are important to avoid manual errors and to provide transparency.	Tool interoperability techniques from GSE viewpoint, Merlin ToolChain.	One case in a large company from consumer-electronics domain. Author has been actively involved in the case (action research).

7.2.2 Second set of industrial cases

In the second phase (Table 15), 16 cases were carried out in order to validate the updated solutions and to evaluate new solutions for existing challenges or to try out solutions for a new set of challenges.

Table 15. Overview of second set of industrial cases.

Topic and GSE viewpoint	Solutions	Company domain and author role
Case 37 Define solutions for early RE phases (gathering, prioritising and recording methods) and gain a better understanding of customers' and other stakeholders' needs. Multiple stakeholders whose viewpoints should be taken into account with the correct weighting, and understood correctly.	Requirements gathering and high-level analysis methods.	One case in a large company in tele-communications domain. Author has been actively involved in the case (action research).
Case 38 Improving integrated tool support from a testing viewpoint in distributed projects. A lot of effort is wasted in GSE on testing, when different/wrong versions are used in replicating defects, or similar tests are run due to not knowing what tests others have performed.	Automated build and test suite. Tool integration platforms and solutions.	One case in a large company from IT services domain. This case is included based on the case experience report.
Case 39 To support integration and verification in collaborative development from the integrator's viewpoint. Integration is the phase where all partners' work comes together and often the problems are identified only then.	Integration and V&V practices in GSE.	One case in a large company in telecommunications domain. This case is included based on the case experience report.
Cases 40 and 41 To improve resource management in distributed development. Optimal use of resources from various sites over several projects is challenging.	Resource management practices and tools.	Two cases in large companies in IT services and electricity network equipment domains. These cases are included based on the case experience reports.
Cases 42 and 43 Improving the sharing of information and knowledge about the on-going projects and other related information in distributed development environments. Information sharing and status monitoring in GSE environments.	Techniques to identify information needs and sources in GSE.	Two cases in a large company in electricity network equipment domain. These cases are included based on the case experience reports.
Cases 44, 45, 46, 47 and 48 Improving tool support for collaborative projects. Tools are important for information sharing in GSE, integrated tools are important to avoid manual errors. The added value of tool integration in GSE is to enable full transparency and traceability of project tasks, requirements, related source codes and build and test status over partners/sites.	Tool interoperability techniques, tool requirements from GSE viewpoint.	Five cases in three large and two SME companies from consumer electronics, IT services, telecommunications, telematic engineering and multimedia content domains.

Topic and GSE viewpoint	Solutions	Company domain and author role
		Author has been actively involved in one of the cases (action research); four cases are included based on experience reports.
Case 49 Improving risk management in distributed projects via surveys. Risk management is complicated due to the distribution; stakeholders have more restricted lines of communication and management by "walking around" is difficult.	Risk management practices and typical risk list for GSE.	One case in a large company in IT services domain. Author has been actively involved in the case (action research).
Case 50, 51 and 52 Improve collaborative development in general.	Identifying challenges and solutions from Prisma wiki.	Three cases in two large and one SME company in electricity network equipment and IT services domains. These cases are included based on the case experience reports.

7.2.3 Summary of the contribution from industrial cases

All of these cases have been documented in a structured way and the experiences have been incorporated in the GSE framework described in sections 3 and 4. The cases have covered the GSE framework areas well as there were cases to all sections of the GSE framework. There were several cases relating to management practices. Five cases (cases 24–28) addressed collaboration management and one case (case 33) addressed IPR management. Five cases (cases 42, 43, 40, 41 and 49) addressed project management issues, two cases, especially from the information sharing viewpoint, one case from the risk management viewpoint and two from resource management viewpoint.

Relating to engineering practices, fifteen cases (cases 1–14 and 37) have addressed various problems in RE, and in addition to that, four cases (cases 19–22) addressed requirements and design documentation. Architectural aspects were addressed in four cases (cases 15–18). Integration and testing practices were addressed in five cases (cases 29–31, 38 and 39), specifically from the tool support point of view in four cases and from an integration practice viewpoint in one case. Release planning was addressed in one case (case 35).

Relating to supporting practices, there was one case addressing defect management (case 32), configuration management (case 34) and improvement processes from the measurement viewpoint (case 23). Six cases (cases 36, 44–48)

addressed integrated tool support for GSE. Also, many of these cases addressed communication issues as part of them.

In addition to these, there were three cases (cases 50–52) that addressed GSE in general, and utilised the developed GSE framework (one case used the Merlin handbook and two cases used the second version of the framework) to address various GSE challenges in the companies concerned. Based on the large number of industrial cases addressing the GSE framework comprehensively, the framework can be seen to be largely based on industrial experience.

8. Reporting the results

This section describes the research performed in the attached publications. The publications bring up novel innovations for global software engineering. Eight publications are included. Their topics are the following:

- I Survey of the state-of-the-practice of collaborative embedded systems' development
- II A more detailed description of the challenges and experience of GSE in a company, namely Philips
- III Overview of the Merlin collaboration handbook: The challenges and solutions in global collaborative product development
- IV Knowledge-related challenges and solutions in GSD
- V Introduction to the RE process in systems' development
- VI A survey of existing RE technologies and their coverage
- VII Experiences in evaluating requirements' quality
- VIII Experiences of tool integration: Development and validation.

Each of the publications discusses topics related to the research questions of this thesis. The first two papers describe the current state of the GSE practices in industry, thus addressing research question 1. The third paper addresses research question 2, and presents the first version of the GSE framework intended to help in finding appropriate solutions for the GSE challenges in a given situation. Paper IV discusses GSE challenges and solutions from a knowledge viewpoint, as knowledge aspects are specifically important in GSE. Papers V–VIII each discuss example solutions to address GSE challenges; Papers V–VII from an RE viewpoint and Paper VIII from a tool-integration viewpoint.

The author has been the main contributor and coordinator of the work reported in Papers I and III–VI. For the work reported in Paper II, the author was responsible for gathering the experience in a structured format, gathering the data via interviews in co-operation with the company representatives, and ensuring the coverage of the relevant topics in GSE. For Paper VII, the author was responsible

for the study of other available methods and application of the LSPCM method for a case involving requirements analysis in a company doing distributed development. For Paper VIII, the author was the coordinator of the work reported in the paper with the specific responsibility of collecting requirements and making decisions about the features to be implemented in the tool chain. Details of the contributions are discussed for each paper in the following sections.

8.1 PAPER I: Collaborative embedded systems development: Survey of state of the practice

Collaborative embedded systems development: Survey of state of the practice sets the basis for the GSE improvement work described in this thesis. The paper was published in the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS) held from March 27–30 2006 in Potsdam, Germany. The research problem of the publication is to present the state of the practice in global embedded software development; that is, to identify the challenges faced by industry in GSE and some solutions available to tackle these challenges. The study presented was based on both industrial surveys (interviews) and a survey of the literature concerning industrial practice. The findings were that the most common collaboration mode was product structure-based subcontracting, while the main motivation for collaboration was most often to save money. However, acquiring competence not available in-house was another common motivation. While the areas that were seen as critical in collaborative product development varied, those most commonly items that were perceived as critical were contracting, change management, requirements development and requirements management. The areas that were most commonly seen as non-critical were software implementation within engineering practices and improvement process and human-resource management within support practices. This survey revealed that the approaches represented by the literature, on the one hand, and industrial practitioners, on the other, towards problems related to collaborative work are different. Industry emphasises technical aspects and detailed problems concerning engineering practices, while the literature focuses on solutions for more general issues such as communication and team building. The findings also included that a lot of solutions are available especially for management and support practices but that only a few solutions could be found for engineering practices.

The author is a co-author of the publication, where Jarkko Hyysalo was responsible for the literature survey part, and the author and Maarit Tihinen for the industrial survey part. The industrial survey part included carrying out the interviews and the analysis of the findings. The author was also the main contributor in the discussion and conclusions of the paper and the coordinator of the inventory work in general.

8.2 PAPER II: Philips experiences of global distributed software development

Philips experiences of global distributed software development discusses a company example of GSE challenges and potential solutions in more detail, providing more in-depth knowledge of the challenges presented in Paper I at a high level. The paper was published in the Empirical Software Engineering Journal, December, 2007.

The experience described comes from more than ten years of global distributed development at Philips, derived from dozens of projects. The main lessons learned were that explicit agreements and ways of working should be defined for the following areas needing the most attention; team coordination and communication, requirements and architectures, integration and configuration management. The main lesson learned from subcontracting software development was the need for explicit attention and ways of working with respect to selection of suppliers, specification of the work to be subcontracted and the establishment and content of the contract.

The author is a co-author of this paper and was responsible for gathering the experience in a structured format, ensuring coverage of the relevant topics in GSE. The author was also responsible for reporting the experiences in publication format, ensuring the background information adequacy. Rob Kommeren (co-author) provided the access to the experience; both his own experience and that of others at Philips. He also provided his contribution to the analysis of the data from the viewpoint of a thorough knowledge of the company.

8.3 PAPER III: Merlin collaboration handbook: Challenges and solutions in global collaborative product development

This paper introduces the Merlin collaboration handbook. The handbook defined the basic structure for GSE improvement and was the first version of the GSE framework. The purpose of the handbook is to support operational collaborative development; that is, to help companies to take care of all the critical aspects during various phases of the collaborative project. In practice, this would be done by collecting and structuring these critical aspects as well as ways to address them, the solutions, into a handbook. The solutions were based on the literature, especially for management and support practices, and on the collection of best practices from Merlin industrial partners via focused interviews on selected topics. Results of these focused interviews were then included as solutions and experiences related to them in the handbook. Finally, the research and development work done during the Merlin project was also added to the handbook as solutions.

The author was the main contributor of this paper and also the coordinator and owner of the Merlin collaboration handbook, which is introduced in this paper. The co-authors Juho Eskeli, Tanja Kynkäänniemi and Maarit Tihinen, each contributed to the handbook development in specific areas and also reported these in this

paper. The paper was published in the Third International Conference on Software and Data Technologies. Porto, Portugal, 5–8 July 2008. Special Session on Global Software Development: Challenges and Advances on ICSOFT 2008.

8.4 PAPER IV: Knowledge related challenges and solutions in GSD

This paper discusses GSE from a knowledge viewpoint, where the key focus is on discussing the challenges that are knowledge intensive and how the knowledge-engineering activities are related to them and, thus, could help in addressing the challenges. This paper mainly brings forward the people aspect in GSE improvement.

A number of knowledge-related challenges may complicate the work in global software development (GSE) projects. In practice, even a small amount of missing knowledge may cause an activity to fail to create and transfer information which is critical to later functions, causing these later functions to fail. Thus, knowledge engineering holds a central role in order to succeed with globally distributed product development. This paper discussed the challenges and solutions based on an extensive literature study and practical experience gained in several international projects over the last decade. The paper analysed the challenges identified in the cases introduced in section 7.2 from a cognitive perspective for bridging and avoiding the knowledge gaps and, based on this analysis, example solutions to address the challenges during the GSE projects were presented. This paper summarises the cases examined from a knowledge-intensive viewpoint.

The author is a co-author of this paper, with the responsibility of analysing the GSE challenges; Maarit Tihinen incorporated the knowledge-engineering viewpoint. The paper was published in Expert Systems, the Journal of Knowledge Engineering in 2011.

8.5 PAPER V: Requirements engineering: Process, methods and techniques

This paper introduces the RE process in general. The paper focuses on RE activities making the distributed nature of the RE clear. The paper was published in a book on RE for Sociotechnical Systems, Eds. Silva, A. & Mate, J., Idea Group, Inc. 2005. The main conclusion of the paper was that RE is a complex process that considers product demands from a vast number of viewpoints, roles, responsibilities and objectives. Thus, RE is generally thought of as the most critical and complex process within the development of embedded systems and it practically always involves some GSE aspects. This paper explains the RE terminology and describes the RE process in detail, with examples of available methods for the main process activities. The main activities described include system requirements development, requirements allocation and flow-down, software requirements development and continuous activities, including requirements documentation, requirements validation and verification and requirements management.

The author is the main author of this paper, and was the coordinator of the work in the project where this paper reports the results. Maarit Tihinen also contributed to the process definition, especially from the available-methods' viewpoint. Marco Lormans and Rini van Solingen contributed to the work, especially from the industrial state-of-the-practice viewpoint.

8.6 PAPER VI: A Survey of existing requirements engineering technologies and their coverage

This paper describes technologies that are available for RE and their shortcomings. These technologies are not specifically analysed from a GSE viewpoint in this paper, but the paper describes RE and the important activities in it that need technology support. This is especially relevant in GSE, as requirements have a pivotal role in sharing work and information within the GSE project. Thus, although the technologies are not GSE specific, they are still very relevant for GSE success, as many of the RE challenges in GSE are related to regular RE practice, such as documenting the requirements clearly, addressing the stakeholders' needs equally or managing the changes properly. These topics are covered by the RE technologies and are presented in the paper. The paper discusses the results of an inventory of the available RE technologies, while also looking into their support in terms of RE.

The author is the main contributor of this paper and coordinator of the work of which this paper is the summary. The methods'-analysis work was shared with Maarit Tihinen, who is the co-author of this paper. The paper was published in the International Journal of Software Engineering and Knowledge Engineering (IJSEKE), 2006.

8.7 PAPER VII: Experiences on evaluating requirements quality

This paper describes example solutions relevant for GSE in more detail. The solutions discussed are related to improving requirements documentation quality. This is specifically relevant in GSE, as the work done at different sites often relies on requirements documentation. Also, as the different backgrounds complicate establishing a common understanding of things it is very important that the descriptions are as clear and unambiguous as possible. The quality of any product depends on the quality of the basis of making it; that is, the quality of the requirements has a strong effect on the quality of the end products. In practice, however, the quality of requirement specifications is poor; in fact, it is a primary reason why so many projects continue to fail. The paper presents a method called LSPCM, developed for certifying software product quality, and describes experiences when using the method for analysing requirements' quality in three cases. The three different cases show that the checks in the LSPCM are useful for finding inconsistencies in

requirements specification, regardless of the application domain. The paper is not specifically written from a GSE viewpoint, but the cases are from projects carried out in distributed settings.

The author is a co-author of this paper that was published in the Third International Conference on Software Engineering Advances. ICSEA 2008. Petra Heck is the developer of the method that the author applied for the case and Petra also documented two cases in the paper. The author studied other available similar methods and applied the LSPCM method for a case that involved requirements analysis in a company doing distributed development.

8.8 PAPER VIII: Experiences of tool integration: Development and validation

This paper describes another example solution relevant for GSE in more detail. The paper discusses tool interoperability, which is very important in GSE, as it can help in sharing information in a timely manner, monitoring status over sites and managing dependencies. The paper was published in the Proceedings of the International Conference on Interoperability of Enterprise, Software and Applications. Berlin, German. March 25–28, 2008.

Generally, in software development, there is a need to link the development-work products with each other; that is, to link the requirements with the corresponding design artefacts to the resulting software and associated test cases. This enables, for instance, efficient change impact analysis and reporting facilities during the different phases of the software development life cycle. Establishing and maintaining these links manually is a laborious and error-prone task, so tool support is needed. This paper describes a configurable tool-integration solution (the Merlin ToolChain) that integrates project management, requirements management, configuration management and the testing of tools. The paper introduces the architecture of the ToolChain as well as describing the development and validation activities that were carried out. Experiences from a real-life industrial case showed that the ToolChain works and is useful in collaborative software development.

The author is a co-author and the coordinator of the work reported in this paper, with the responsibility of collecting the requirements and making decisions about the features. Jukka-Pekka Pesola and Juho Eskeli were responsible for implementing the ToolChain and writing the implementation description in the paper. Rob Kommeren and Martin Gramza represented Philips and provided the evaluation environment and experience for the ToolChain trial.

9. Discussion

In this section the results of the work are evaluated according to the research questions. Then the validity of the research is discussed.

9.1 Evaluation of the results

The main contribution of this work lies in providing the first comprehensive framework addressing the critical issues in GSE. There are many publications about GSE, but they usually focus on some specific topic or on a single company experience. This thesis brings together these publications as well as many other cases from industry, structured into a framework that helps to comprehensively analyse challenges and thus identify relevant solutions for the various GSE situations.

Relating to the first research question:

- What are the main challenges faced by companies when doing GSE? How can these challenges be categorised?

There are multiple challenges that companies face during GSE projects, but similarities could be identified in the challenges and root causes behind the challenges could be traced to three main groups: basic GSE circumstances (time difference and distance, multiple partners/stakeholders), derivative GSE causes (lack of communication, coordination breakdown and different backgrounds and tacit knowledge) and consequent GSE causes (lack of teamness and trust). Also, a framework for identifying challenges covering management, engineering and supporting practices was described. The main challenges faced by companies vary, as was discussed in section 3.1.

The second research question was:

- Are there solutions to address these challenges? How can these solutions be categorised in order to support finding the suitable ones for each situation?

For each of the challenges identified in industry, solutions were developed or identified and adapted from the literature. These solutions were presented using three dimensions: process, technology and people and the basic structure of the

GSE framework. This enables the finding of solutions to the relevant challenges for each case. These solutions have been validated in industrial cases and action research and made available in the SameRoomSpirit Wiki. In total, in the current version of the wiki, the solutions are based on more than 130 published scientific articles. The solution descriptions are based purely on the industrial partners' experience (27%), purely on literature (46%) and on a combination of both experience and literature (27%). However, the solutions are partly overlapping; that is, separate solutions can have similar topics, and thus, more than 30% of the solutions are addressed both in the literature as well as in the industrial cases.

The third research question was:

- What are the critical activities/subprocesses in GSE? Are there solutions that support implementing these critical subprocesses?

The most critical activities/subprocesses were identified and discussed in section 3.1. RE was chosen as an example of a critical subprocess because it was identified as one of the most critical ones by industry and also by other researchers. RE challenges and solutions were identified by utilising the developed framework as an example of how GSE practices can be improved in practice. Also, several cases were examined relating to RE in GSE.

9.2 Validity of the research

The research methods used in the thesis were a literature study, case studies and action research. According to Easterbrook et al. (2007), the major weakness of case studies is that the data collection and analysis is more open to interpretation and researcher bias. For this reason, an explicit framework is needed for selecting cases and collecting data. Although an individual case study often reveals deep insights, the validity of the results depends on a broader framework of empirical induction. For example, in confirmatory case studies, evidence builds when subsequent case studies also support the theory and/or fail to support rival theories. To address this challenge, several industrial cases were carried out, both by the author and by other researchers, and the results from these cases were used together. The results were also then reviewed by industrial representatives and other researchers. Also, an established and commonly accepted framework, CMMI was used as a basis to develop the structure of the GSE framework.

Commonly used criteria to evaluate the validity of research (Easterbrook et al., 2007; Runeson & Höst, 2009) include construct validity, internal validity, external validity and reliability. *Construct validity* focuses on whether the theoretical constructs are interpreted and measured correctly. Problems with construct validity occur when the measured variables do not correspond to the intended meanings of the theoretical terms. If, for example, the constructs discussed in the interview questions are not interpreted in the same way by the researcher and the interviewed persons, there is a threat to the construct validity. Means to improve construct validity (Yin, 2003) include, for example, multiple sources of evidence and

having key informants review draft case-study reports. The research covered several case studies in several companies, as well as experience reports published by others, thus having multiple sources of evidence. Also, all the interview reports and case reports were reviewed by the company representatives. Also, the first complete framework, the Merlin handbook, was evaluated by 16 external testers who used the handbook and provided feedback and secondly the handbook was used in an industrial case to support improving subcontracting efficiency in a company.

Internal validity focuses on the study design and particularly on whether the results really do follow from the data. When the researcher is investigating whether one factor affects an investigated factor there is a risk that the investigated factor is also affected by a third factor. If the researcher is not aware of the third factor and/or does not know to what extent it affects the investigated factor, there is a threat to the internal validity. As the cases involved real-life projects and not laboratory settings, the factors affecting the results cannot be isolated. Due to that the research goal was to understand the GSE and its challenges and solutions in practice the real life industrial projects were seen as necessary. However, the internal validity has been supported in that the results have been analysed and validated by several persons: the author, company representative, external reviewers and the case participants. The company representatives and case participants have evaluated the impact of the tried practices in the cases. This increases the internal validity, as they were independent of the author and had a good understanding of the possible other factors that may have affected the case. In case of those factors, they have also been described in the experience reports and taken into account in the GSE framework. Also, similar results have been found in several cases that have been carried out by several different persons within the Merlin and Prisma projects as well as in independent companies that have been published in research papers.

External validity focuses on whether claims for the generality of the results are justified. Often, this depends on the nature of the sampling used in a study. During analysis of external validity, the researcher tries to analyse to what extent the findings are of relevance for other cases. This is supported by the multiple case studies on similar topics in several companies. Also, in the two phases of industrial cases, the second set of cases partly validated the results of the first set of cases by addressing the same or similar challenges and by evaluating the framework documented in the SameRoomSpirit Wiki.

Reliability focuses on whether the study yields the same results if other researchers replicate it. Problems occur if the researcher introduces bias, perhaps because the tool being evaluated is one that the researcher herself has a stake in. The case studies included in this research were also carried by other researchers than the author, providing evidence of reliability.

According to Easterbrook et al. (2007), two key criteria for judging the quality of action research are whether the original problem is authentic (i.e., whether it is a real and important problem that needs solving) and whether there are authentic knowledge outcomes for the participants. Relating to the authenticity, the problems

addressed in the industrial work have been defined by the companies themselves, not by the author. Relating to the authenticity of the knowledge, the work was carried out in real industrial projects, and the conclusions were reached in co-operation with the industrial representatives and were also reviewed by more members from the companies involved. Also, the results documented in the GSE framework versions have been reviewed by external company representatives and researchers. Action research is also characterised by a commitment to effect real change and by an iterative approach to problem solving. The biggest challenge for action research is its immaturity as an empirical method. Although frameworks for evaluating action research have been proposed (e.g., Lau, 1999), they tend to be vague or subjective, leading to accusations that action research is ad hoc (Easterbrook et al., 2007). Because of this, the action research was carried out in combination with the case studies, thus, the action research results can be claimed to be valid.

The used methods are primarily qualitative and these methods rely on field-work, using techniques such as participant observation and interviews. Key challenges include preparing good questions for structured or semi-structured interviews and finding the time and resources needed to collect and analyse potentially large sets of data (Easterbrook et al., 2007). Furthermore, as stated by Easterbrook et al. (2007), all research conducted in industrial settings brings a number of challenges. It can be very hard to gather data to find out what practitioners actually do or what needs to be improved in the organisation, rather than what practitioners say they do or what they think requires improvement. In return for access to the organisation, the researcher usually has to give up some control. For example, it is hard to observe and document findings without interfering with the observed situation, especially when the industrial partners want to know in advance what the expected outcomes are. It is often difficult to know if changes are made through involvement in the research or if they would have occurred anyway.

As a conclusion, the main weakness of the work is that the topic of the thesis is so vast that it is not possible to cover all of the issues in great detail. However, the strength of the work is in the large number of industrial cases that the author has participated in at various levels and that are also supported by the experience reported in publications by others. Also, a more specific area, namely RE, was addressed in more detail.

10. Summary and conclusions

This thesis summarises and extends eight original publications about challenges and solutions for GSE. The main contribution of this work lies in providing the first comprehensive framework addressing the critical issues in GSE from management, engineering and supporting practices viewpoints. There are many publications about GSE, but they usually focus on some specific topic or on a single company experience. This thesis brings together these publications as well as many other cases from industry, structured into a framework that helps to comprehensively analyse challenges and thus identify relevant solutions for the various GSE situations.

GSE means software engineering that is carried out in globally distributed settings at various geographical locations. The work can be done either within a company (multi-site development) or in collaboration between companies at different locations. Under the current market pressures, companies need to use their existing resources as effectively as possible and they also need to utilise the global resource and expertise pool. This has resulted in GSE becoming increasingly common and the ability to collaborate effectively has become a critical factor in the software development life cycle. The main expected benefits from GSE are improvements in time-to-market efficiency, being close to the customers and having flexible access to greater and less costly resources. However, there are also a number of problems, which lead to the full benefits of GSE not being reached. This means that productivity in a distributed project can drop by up to 50 per cent, with rework of two to five times more than for a collocated project. The thesis discussed various collaboration modes and its main focus was on customer–supplier relations and in-house distributed development, as those have been the most commonly used models in the case organisations and in the available related work.

The thesis analysed the main challenges faced by the companies when doing GSE, and identified solutions to the challenges from people, process and technology viewpoints. RE in GSE was discussed as an example of how GSE affects a critical subprocess. The research methods in this thesis involved a literature study to define the basis and to gather other researchers' views on the topic, and case studies and action research to investigate real-life situations in companies. Based on the empirical work, the most challenging areas were identified to be contracting

and requirements definition, project planning and tracking, architecture analysis/design, the integration phase (testing) and co-operative work in general.

The challenges were then discussed based on their root causes, using three levels – basic GSE circumstances, derivative GSE causes and consequent causes – and then summarised in a table using the GSE framework structure. Then GSE solutions were discussed from people, process and technology viewpoints and summarised again into the GSE framework. As an example, RE was first discussed in general and then the challenges were discussed as presented in the literature. Then the RE challenges were discussed and elaborated based on the empirical work according to the same structure; that is, as basic GSE circumstances, derivative GSD causes and consequent causes and then summarised in a table. Then solutions for RE challenges were discussed from process, people and technology viewpoints. The work reported in this thesis is based on extensive empirical work, carried out over several years. The empirical work was carried out in several phases: in the first phase, an industrial inventory, including industrial experience reported by others in the literature, was studied. Based on this, an initial framework for GSE was developed, including the main challenges to be addressed in GSE projects and the solutions for them. After this, two sets of industrial cases were examined, addressing a wide set of GSE aspects and challenges. Altogether, 52 industrial cases were examined during the projects over the years 2004–2011 relating to distributed development.

The topic of the thesis is very wide, so the GSE framework does not necessarily cover all the challenging aspects of GSE. However, it covers things that have been seen as important in most of the case companies. In addition to the cases, the framework has also been validated via several reviews and feedback from other researchers and practitioners. Thus, it can be claimed that the topics addressed in the GSE framework are the topics that should be addressed in the companies in order to succeed with GSE.

Currently, GSE is a fact of life in software-intensive systems development and it will become even more common, making it rather an exception to work at a single site. This thesis has shown clearly that GSE is challenging and companies need to realise that the savings gained by cheaper hourly rates are not direct savings for the company, as there are usually overheads that are caused by the distribution. This should be taken into account when making the decisions to distribute work. The potential benefits of GSE are, however, high; it can enable a company to remain competitive in the ever-tougher markets by enabling a focus on the core competences and on innovating new products and business instead of spending time on simpler tasks. The work presented in this thesis is a step towards better, more productive and higher quality GSE. In the future the GSE framework could be extended to a phased model, which would describe the most essential activities and the activities to be done next in order to enable companies to improve their practice step by step. Also, the emerging collaboration modes, such as joint ventures or utilising open communities in commercial product development, need more attention. They were not addressed much in this thesis, as the case companies were not yet utilising these models.

References

Ahmad, N. & Laplante, P. A. 2006. Software project management tools: Making a practical decision using AHP. In: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, Columbia, Maryland, USA, pp. 76–84. ISBN 0-7695-2624-1, DOI 10.1109/SEW.2006.30.

Al-Ani, B. & Redmiles, D. 2009. Trust in distributed teams: Support through continuous coordination. IEEE Software, Vol. 26, No. 6, pp. 35–40. DOI 10.1109/MS.2009.192.

Baskerville, R.L., 1997, Distinguishing action research from participative case studies, Journal of Systems and Information Technology, Vol. 1, No. 1, pp. 25–45. DOI 10.1108/13287269780000733.

Battin, R., Crocker, R., Kreidler, J. & Subramanian, K. 2001. Leveraging Resources in Global Software Development, IEEE Software, Vol. 18, No. 2, March/April 2001, pp. 70–77. DOI 10.1109/52.914750.

Bhat, J. M., Gupta, M. & Murthy, S., N. 2006. Overcoming requirements engineering challenges: Lessons from offshore outsourcing. IEEE Software, Vol. 23, No. 5, September/October 2006, pp. 38–44. DOI 10.1109/MS.2006.137.

Borchers, G. 2003. The software engineering impacts of cultural factors on multicultural software development teams. In: Proceedings of the 25th International Conference on Software Engineering (ICSE'03), IEEE, pp. 540–545. ISBN 0-7695-1877-X, DOI 10.1109/ICSE.2003.1201234.

Brockmann, P. S. & Thaumüller, T. 2009. Cultural aspects of global requirements engineering: An empirical Chinese-German case study. In: Proceedings of the fourth IEEE International Conference on Global Software Engineering, Limerick, Ireland, 13–16 July 2009, pp. 353–357. ISBN 978-0-7695-3710-8, DOI 10.1109/ICGSE.2009.55.

Carmel, E. 1999. Global software teams: Collaborating across borders and time zones. Prentice-Hall, Upper Saddle River, N.J. ISBN-13: 978-0139242182.

Carmel, E. & Tija, P. 2005. Offshoring information technology: Sourcing and outsourcing to a global workforce. Cambridge University Press. ISBN-13 978-0521843553.

Casey, V. & Richardson, I. 2008. Virtual teams: Understanding the impact of fear. Software process improvement and practice, Vol. 13, Issue 6, pp. 511–526. DOI: 10.1002/spip.404.

Cheng, B. H. & Atlee, J. M. 2007. Research directions in requirements engineering. In: Future of Software Engineering, 2007. FOSE '07, 23–25 May 2007, pp. 285–303. ISBN 0-7695-2829-5, DOI 10.1109/FOSE.2007.17.

CMMI. 2006. CMMI for development, version 1.2., Technical Report CMU/SEI-2006-TR-008.

da Silva, F.Q.B., Costa, C., Frana, A.C.C. & Prikladinicki, R. 2010. Challenges and solutions in distributed software development project management: A systematic literature review. In: Global Software Engineering (ICGSE) 2010. 5th IEEE International Conference on Global Software Engineering, Recife, Brazil, pp. 87–96. ISBN 978-1-4244-7619-0, DOI 10.1109/ICGSE.2010.18.

Damian, D. 2002. The study of requirements engineering in global software development: as challenging as important. In: Proceedings of Global Software Development, Workshop #9. Organized in the International Conference on Software Engineering (ICSE) 2002, Orlando, FL, ISBN 1-86365-699-5.

Damian, D., Lanubile, F., Hargreaves, E. & Chisan, J. 2004. The 3rd international workshop on global software development. In: Proceedings of ICSE 2004. International Conference on Software Engineering, Edinburgh, Scotland, May 2004, pp. 756–757. ISBN 0-7695-2163-0, DOI 10.1109/ICSE.2004.1317521.

Damian, D. E. & Zowghi, D. 2002. An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In: Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03). ISBN 0-7695-1874-5, DOI 10.1109/HICSS.2003.1173665.

Damian, D. & Zowghi, D. 2003. Requirements engineering challenges in multi-site software development organizations. Requirements Engineering Journal, Vol. 8, No. 3, pp. 149–160. DOI: 10.1007/s00766-003-0173-1.

Davison, R. M., Martinsons, M. G., & Kock, N., 2004, Principles of Canonical Action Research, Information Systems Journal, Vol. 14, No. 1, pp. 65–86, DOI DOI: 10.1111/j.1365-2575.2004.00162.x.

Duysters G. & Hagedoorn J. 2000. A note on organizational modes of strategic technology partnering. Journal of Scientific and Industrial Research, Vol. 58, August/September 2000, pp. 640–649.

Easterbrook, S. M., Singer, J., Storey, M. & Damian, D. 2007. Selecting empirical methods for software engineering research. In: Shull, F. & Singer, J., (Eds.). Guide to Advanced Empirical Software Engineering. Springer, 2007, pp. 285–311. DOI 10.1007/978-1-84800-044-5_11.

Ebert, C. & De Neve, P. 2001. Surviving global software development. IEEE Software, Vol. 18, No. 2, March/April 2001, pp. 62–69. DOI 10.1109/52.914748.

Ebert, C., Murthy, B. K. & Jha, N. N. 2008. Managing risks in global software engineering: Principles and practices. In: Proceedings of the IEEE International Conference on Global Software Engineering, 17–20 August 2008, pp. 131–140. ISBN 978-0-7695-3280-6, DOI 10.1109/ICGSE.2008.12.

Faems, D. 2003. Linking technological innovation and inter-organizational collaboration: An overview of major findings. Working Paper Steunpunt OOI: March 2003. http://www.ondernemerschap.be/Upload/Documents/STOOI/Working%20Papers/2003/linking_techn_innovation.pdf. Available 19 March 2012.

Firesmith, D. G. 2005. Quality requirements checklist. Journal of Object Technology, Vol. 4, No. 9, November–December 2005, pp. 31–38. http://www.jot.fm/issues/issue_2005_11/column4, Available 19 March 2012.

Forrester 2010. Making collaboration work for the 21st century's distributed workforce, White paper, Forrester Research Inc., November 2010.

Fryer, K. & Gothe, M. 2008. Global software development and delivery: Trends and challenges, 15 January 2008, White paper. http://www.ibm.com/developerworks/rational/library/edge/08/jan08/fryer_gothe/index.html. Available 30 March 2012.

Gotel, O. & Finkelstein, A. 1994. An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, 18–22 April 1994, pp. 94–101. ISBN 0-8186-5480-5, DOI 10.1109/ICRE.1994.292398.

Graaf, B., Lormans, M. & Toetenel, H. 2003. Embedded software engineering: state of the practice. IEEE Software Magazine, Vol. 20, No. 6, pp. 61–69. DOI 10.1109/MS.2003.1241368.

Grinter, R. E., Herbsleb, J. D. & Perry, D. E. 1999. The geography of coordination: Dealing with distance in R&D work. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, 1999, pp. 306–315. ISBN 1-58113-065-1, DOI 10.1145/320297.320333.

Hagedoorn, J. 2000. Inter-firm R&D partnerships – An overview of major trends and patterns since 1960. Research Policy, Vol. 31, Issue 4, May 2002, pp. 477–492. DOI 10.1016/S0048-7333(01)00120-2.

Hanisch, J. & Corbitt, B. 2004. Requirements engineering during global software development: Some impediments to the requirements engineering process – a case study. In: Leino T., Saarinen T. & Klein, S. (Eds.). Proceedings of the Twelfth European Conference on Information Systems, pp. 628–640. ISBN 951-564-192-6.

Heindl, M. & Biffl, S. 2006. Effective risk management with tracing the rationale of software requirements in highly distributed projects. Proc. of the Global Software Development Workshop at the Int. Conf. on Software Engineering, Shanghai, China, May 2006. ISBN 1-59593-404-9, DOI 10.1145/1138506.1138512.

Herbsleb, J. & Mockus, A. 2003. An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering, Vol. 29, No. 6, June 2003, pp. 481–494. DOI 10.1109/TSE.2003.1205177.

Herbsleb, J., Mockus, A., Finholt, T. & Grinter, R. 2001. An empirical study of global software development: Distance and speed. In: Proceedings of the International Conference on Software Engineering, 2001, Toronto, Canada, May 15–18, pp. 81–90. ISBN 0-7695-1050-7, DOI 10.1109/ICSE.2001.919083.

Herbsleb, J. & Moitra, D. 2001. Global software development. IEEE Software, Vol. 18, No. 2, March/April 2001, pp. 16–20. DOI 10.1109/52.914732.

Hofstede, G. 2001. Culture's consequences. Comparing Values, Behaviors, Institutions, and Organizations Across Nations. Sage Publications, London. 2nd edition. ISBN-13: 978-0803973244.

Holmstrom, H., Conchuir, E. O., Ågerfalk, P. J. & Fitzgerald, B. 2006. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In: Proceedings of IEEE International Conference on Global Software Engineering (ICGSE'06), October 2006, IEEE, pp. 3–11. ISBN 0-7695-2663-2, DOI 10.1109/ICGSE.2006.261210.

Hossain, E., Ali Babar, M. & June Verner, J. 2009. How can agile practices minimize global software development co-ordination risks? In: Software Process Improvement, Communications in Computer and Information Science. Vol. 42, Part 3, pp. 81–92. ISBN 978-3-642-04132-7.

IEEE 1990. Software Engineering. IEEE Standard 610.12-1990. IEEE Standard Collection. DOI 10.1109/IEEESTD.1990.101064.

Juristo, N., Moreno, A. M. & Silva, A. A. 2002. Is the European industry moving toward solving requirements engineering problems? IEEE Software, Vol. 19, No. 6, pp. 70–77. DOI 10.1109/MS.2002.1049395.

Kanstren, T., Kääriäinen, J., Soininen, S., Takalo, J., Heinonen, S. & Teppola, S. 2007. Merlin White paper: Technical implementations: Tool analysis. http://virtual.vtt.fi/virtual/proj1/projects/merlin/handbook/merlinhandbook_white%20paper_technicalimplementationstoolanalysis_final.pdf. Available 30 March 2012.

Kotonya, G. & Sommerville, I. 1998. Requirements engineering: Process and techniques. John Wiley & Sons. ISBN-13: 978-0471972082.

Lau, F. 1999. Toward a framework for action research in information systems studies. Information Technology & People, Vol. 12, No. 2, pp. 148–176. DOI 10.1108/09593849910267206.

Lindström, M. 2003. Ensuring availability and access to new and existing technologies in cellular terminal business. Dissertation. Helsinki University of Technology. ISBN 951-22-6521-4.

López, A., Nicolás, J. & Toval, A. 2009. Risks and safeguards for the requirements engineering process in global software development. In: Proceedings of the fourth IEEE International Conference on Global Software Engineering, Limerick, Ireland, 13–16 July 2009, pp. 394–399. ISBN 978-0-7695-3710-8, DOI 10.1109/ICGSE.2009.62.

Martin, J. 1984. An Information systems manifesto. Prentice Hall, ISBN-13: 978-0134647692.

Merlin, 2004–2007. ITEA project, Embedded Systems Engineering in Collaboration. http://virtual.vtt.fi/virtual/proj1/projects/merlin/index.html. Available 15 July 2011.

Mockus, A. & Hersleb, J. 2001. Challenges of global software development. In: Proceedings of 7th International Software Metrics Symposium, 4–6 April 2001, pp. 182–184. ISBN 0-7695-1043-4, DOI 10.1109/METRIC.2001.915526.

Molli, P., Skaf-Molli, H., Oster, G. & Jourdain, S. 2002. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. The 7th International Conference on Computer Supported Cooperative Work in Design, 2002, IEEE, pp. 80–84. ISBN 85-285-0050-0, DOI 10.1109/CSCWD.2002.1047653.

Nonaka, I. & Takeuchi, H. 1995. The knowledge-creating company. Oxford University Press, New York, USA. ISBN-13: 978-0195092691.

Nuseibeh, N. B. & Easterbrook, S. 2000. Requirements engineering: a roadmap. In: Proceedings of the IEEE International Conference on Software Engineering (ICSE), pp. 35–46. ISBN 1-58113-253-0, DOI 10.1145/336512.336523.

Paasivaara, M. & Lassenius, C. 2004. Collaboration practices in global inter-organizational software development projects. Software Process Improvement and Practice, Vol. 8, No. 4, pp. 183–199. DOI 10.1002/spip.187.

Poston, R. M. & Sexton, M. P. 1992. Evaluating and selecting testing tools. IEEE Software, Vol. 9, No. 3, pp. 33–42. DOI 10.1109/52.136165.

Prisma 2009–2011. ITEA2 project. Productivity in Collaborative Systems Development. URL: http://www.prisma-itea.org (Accessed 19 March 2012).

Prisma 2011. The same room spirit wiki. http://www.sameroomspirit.org. (Accessed 19 March 2012).

Revelle, J. B., Moran, J. W. & Cox, C. A. 1998.The QFD Handbook. John Wiley & Sons. ISBN-13: 978-0471173816.

Robertson, S. & Robertson, J. 1999, Mastering the requirements process. Addison-Wesley, ISBN-13: 978-0201360462

Runeson, P. & Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14, pp. 131–164, DOI 10.1007/s10664-008-9102-8.

Seppänen, V., Helander, N., Niemela, E. & Komi-Sirviö, S. 2001. Original software component manufacturing: survey of the State-of-the-Practice. In: Proceedings of the 27th Euromicro Conference, 2001, pp. 138–145. ISBN 0-7695-1236-4, DOI 10.1109/EURMIC.2001.952448.

Simons, M. 2006. Distributed agile development and the death of distance. Sourcing and Vendor Relationships Advisory Service, Executive Report, Vol. 5, No. 4. Arlington, MA, USA, Cutter Consortium.

Sommerville, I. & Sawyer, P. 1997. Requirements engineering: A good practice guide. John Wiley & Sons. ISBN-13: 978-0471974444.

Standish Group 2006. The CHAOS Report, published on http://www.standishgroup.com 1996, 1998, 2000, 2002, 2004 and 2006. The Standish Group International, Inc.

Susman, G.I. and Evered, R.D., 1978, An assessment of the scientific merits of action research, Administrative Science Quarterly, Vol. 23, No. 4, pp. 582–603. Article Stable URL: http://www.jstor.org/stable/2392581.

Sutcliffe, A. 1998. Scenario-based requirement analysis. Requirements Engineering Journal, Vol. 3, No. 1, pp. 48–65. DOI 10.1007/BF02802920.

Thompson, I. 2001. Collaboration in technical communication: A qualitative content analysis of journal articles, 1990–1999. IEEE Transactions on Professional Communication, Vol. 44, No. 3, September 2001. DOI 10.1109/47.946462.

Trompenaars, F. & Hampden-Turner, C. 1997. Riding the waves of culture. 2nd edition. McGraw-Hill. ISBN-13: 978-1857881769.

Välimäki, A., Kääriäinen, J. & Koskimies, K. 2009. Global software development patterns for project management. In: Proceedings of EuroSPI 2009, Springer, September 2009, pp. 137–148, DOI 10.1007/978-3-642-04133-4_12.

Wahyudin, D. M., Heindl, S., Biffl, A. & Schatten, B. R. 2007. In-time project status notification for all team members in global software development as part of their work environments. In: Proceeding of SOFPIT Workshop 2007, SOFPIT/ICGSE, Munich, pp. 20–25.

Welborn, R. & Kasten, V. 2003. The Jericho Principle, how companies use strategic collaboration to find new sources of value. John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN-13: 978-0471327721.

Williamson, O. E. 1996. The mechanisms of governance. Oxford, Oxford University Press. ISBN-13: 978-0195132601.

Winkler, D., Biffl, S. & Kaltenbach, A. 2010. Evaluating tools that support pair programming in a distributed engineering environment. In: Proc. Conference on Evaluation and Assessment in Software Engineering (EASE), Keele,

Great Britain, 12–13 April 2010, pp. 1–10. http://ewic.bcs.org/content/ConWebDoc/34785. Available 19 March 2012.

Yin, R. K. 2003. Case study research: Design and methods. 3rd edition. Sage, Newbury Park, CA, USA. October 2008. ISBN-13: 9780761925521.

Yousuf, F., Zaman, Z. & Ikram, N. 2008. Requirements validation techniques in GSD: A survey. In: Proceedings of Multitopic Conference, 2008. INMIC 2008. IEEE International, Publication Date: 23–24 December 2008, pp. 553–557. ISBN 978-1-4244-2823-6, DOI, 10.1109/INMIC.2008.477780.

Appendix A: GSE interview framework

This appendix describes industrial inventory framework for the Merlin project. This framework intends to cover company collaboration widely, thus all topics in the framework may not be addressed with every company. Topics to be addressed are selected in co-operation with the industrial partner and VTT, based on industrial partner's interests and coverage of the framework in general, meaning, that some additional topics may be addressed with a company so that the whole framework is covered at least by one company inventory. In any case, all topics that are seen important by the company are addressed.

With every topic, also discuss, is the current way of working good or should it be improved and reasoning for that. Also, improvement ideas should be discussed.

1. General

- What kind of collaboration modes does your organisation currently have, or you have been involved with?
- Are there differences in importance and effect in different collaboration modes?
- What is the general motivation for collaboration (to save money, to save time, to focus on own competence areas, to share risks, to enhance existing products, to develop new products, etc.)?
- Is information exchange planned and documented (e.g. information flows between collaborative partners)? Is this important?
- Who selects the collaboration mode (project level, organization wide, etc.)? How and in what level?
- Who is responsible of defining and improving common practices between collaboration partners?

2. Management practices

- How do organisational changes affect collaboration?
- How is the impact of organisational changes mitigated?

2.1 Collaboration strategy

- Do you have a specific (sub)strategy for company collaboration?
- What kind of impact does the strategy have on different collaboration modes (joint research, subcontracting, technology exchange)?

- How is the company collaboration visible in business models?
- Does business model implicitly include co-operation with other companies as part of business model?
- What kind of impact does the business model have on different collaboration modes (joint research, subcontracting, technology exchange)?
- What options are considered during make/buy analysis (make yourself, buy COTS component, buy development work from supplier)?
- What analysis methods or techniques are used during make/buy analysis?
- What kind of information is available (concerning the product and suppliers) during make/buy analysis?
- What are the main criteria used in the make or buy analysis?
- What kind of information is available concerning the suppliers for supplier selection?
- How are the suppliers selected?
- What are the main criteria for supplier selection?
- What kinds of methods/techniques/tools are used for supplier selection?
- Are the customers prioritised, how?
- What are the main criteria for customer priorisation?
- What kinds of methods/techniques/tools are used for customer priorisation?
- Are company dependencies identified? How strong are the dependencies (importance of selected customers/suppliers to company)?
- What kind of alternative strategies/customers/suppliers exist?
- What kinds of mitigation strategies exist to handle company dependency?
- Technology refreshment:
  - Are products planned to be refreshed?
  - What kinds of technology refreshment strategies do you have?

2.2 Contracts

- How are the intellectual property rights (IPRs) handled in the contracts?
- What kind of contract is established? Do you use legal counseling to help with contracts?
- Who owns the developed product/code?

- Do you have any long-term agreements with suppliers or customers? Do any of these contracts/agreements have built in price adjustments?
- Does contracting process efficiency affect collaboration?
- Do you incentivize software quality (in addition to schedule and cost)?

2.3 Project management process

- How are collaboration modes taken into account in project scheduling (dependencies to supplier, requirements from customers)?
- How are collaboration modes taken into account in project efforts estimation (saved effort form using existing material (COTS), additional effort for development of glueware)? And human resource planning?
- How are collaboration modes taken into account in project costs estimation?
- How and what information and material is shared during project planning? What information is needed from supplier/customer for project planning?
- Are practices, formats and responsibilities for exchanging information between partners defined and agreed between partners? Is this important?
- Is there a need to have specific roles for collaboration management?
- What collaboration practices need to be agreed with parties?
- How is project plan approved between collaboration parties?
- How is the progress of product development tracked during the project (what kind of meetings, reports etc.)?
- Who is responsible of tracking project in joint projects?
- How is project tracking done between collaboration parties? (Are plans/estimates compared to actuals, what happens if there are deviations, are identified problems tracked?)
- How are changes to project (schedule, scope, people) managed and communicated?
- Is there control over changes made by other collaboration parties (e.g., from acquirer to COTS provider release schedule)?
- What authority is needed?

2.4 Risk management

- How do risk management and analysis differ between collaboration and in-house development?

  - Are there risks that are caused by work done in collaborative way?
- Can monitoring or controlling the collaboration (e.g., metrics) support risk management?
  - Do you continuously follow and update collaboration risks?

2.5 Collaboration management

- How is communication between collaboration parties organised? Are responsibilities defined? Are following roles named or part of some other role:
  - Subcontract managers
  - Collaborator coordinator
  - Product integrator (person)?
- How is interface with collaboration parties work defined and agreed?

Customer–subcontractor relationship

- How is the subcontractor/customer management organised?
  - Do you have designated person(s) that are responsible of communications to the subcontractor?
- How are subcontractors selected? What kind of information is available regarding them?
- What are the main criteria for subcontractor selection?
- What kinds of methods/techniques/tools exist for subcontractor selection?
- How is the work to be subcontracted defined and planned?
- Are periodic technical interchanges held with subcontractors?
- Are the results and performance of the software subcontractor tracked against their commitments?
- Who does the tracking (roles and responsibilities)?
- How are changes to subcontractors work agreed?
- How are subcontracted work products accepted?
- Are subcontractor assessments done, what topics are included, what methods, techniques or tools are used?
- Do you evaluate software capability of subcontractors (SPICE, CMMI)?
- Do you evaluate the software process proposed/practiced by the subcontractor in subcontracting?
- Do you share effort and cost information (direct materials, direct labor etc.)?

- How is the customer support organized?
- What kind of feedback is received from the customer?
- Do you integrate system acquisition and software acquisition?
- Do you know software capability of subcontractor's teams?
- What kinds of estimates are defined for the subcontractors (size, effort, cost, schedule)?
- Which deliverables are required from the subcontractors (plans, requirements, architecture, test plans, metrics, maintenance)?
- What kind of criteria is used for selection open source software (documentation, support)?

Customer–supplier (licensor) relationship

- Who keeps track of existing/acquired COTS, open source, suppliers etc.?
- How are suppliers selected? What kind of information is available regarding the suppliers?
- What are the main criteria for supplier selection?
- What kinds of methods/techniques/tools exist for supplier selection?
- Do you have named person(s) that are responsible for communications to the supplier?
- What kinds of processes or guidelines exist for customer acceptance?
- How have authorization and responsibilities been agreed with the supplier?
- Do you analyse and follow-up financial status of the suppliers?
- Are supplier assessments done, what topics are included?
- What kind of methods, techniques or tools do you use in supplier assessment?
- How do you rate your suppliers?
- Do you share effort and cost information (direct materials, direct labor etc.)?
- How is the customer support organized?
- What kind of feedback is received form the customer?
- Do you have a specific contract for customer support or is that part of development agreement?

Joint development relationship

- Do you have named person(s) that are responsible for communications to the partner?

- How are partners selected?

2.6 Quality management

- Do the quality goals differ between in-house and collaboration development? What kind of differences can be recognised?
- Quality monitoring: How can "quality" be monitored/controlled during the collaborative development (e.g., quality of work products, errors, bugs, delays)?
  - Are some quality issues (or development phases) more critical than others in collaboration project? How are they different from in-house development, and why?
  - What kinds of metrics can be defined for monitoring quality of work products (or progressing of the development process)?

3. Engineering practices

- When is the decision made on which product parts will be developed externally and what are critical competence to the company and should be made in-house?
- When are make-or-buy decisions made, what technical reasoning is used?
- How are the different collaboration modes taken into account in release strategy?
- How are responsibilities shared and agreed in joint product development?
- Are collaboration decisions and rationale documented?
- What should you know before you can decide collaboration mode?
- Do you have some framework for selecting licensed (COTS, OS) components?

3.1 Requirements development

Requirements gathering

- How are the collaboration parties involved in requirements gathering?
  - Are requirements gathered also from the involved collaboration parties (in addition to in-house)?
  - Are all gathered requirements shared with collaboration parties? What information is shared?
  - How are requirements gathered from customers (by COTS provider)?

High level analysis of system requirements (incl. prioritising requirements)

- Who does the high level analysis in different collaboration modes? Are responsibilities and authority clear?
- How are the requirements, proposed solution, and their relationships communicated to all affected parties (specific description technique etc.)?
- How can the common understanding be ensured?
- When is applicability of the existing (licensed) parts analysed against requirements?
- Can possibilities of acquired material be taken into account (change requirements so that something “almost fitting” can be used)?
- Do priorities of requirements affect make-or-buy decisions or selection of collaboration mode?
- How are the effects of using licensed parts analysed?
- How do the licensed parts affect high-level analysis (priorities)?
- Are the produced system requirements validated? How, who participates?
- Is the (to be) acquired software taken into account when analysing system requirements?

Detailed analysis of requirements

- Who does the detailed analysis in different collaboration modes?
- How (who, based on what) is this done with licensed components? What documentation is needed?
- Are uniform documentation practices needed between collaboration parties?
- How are requirements validated?

Allocation and flow-down

- Should the solution already at this level include also descriptions of what will be developed in-house and what not?
- How are non-functional requirements allocated to the parties?
- How can it be ensured that, e.g., the latest versions of design or requirements documents are available for each sub-system? Especially, if changes have been done in some sub-system that will have influence on other sub-systems?
- Should the interfaces between parts developed by the parties be defined and in what detail?
- Is needed wrapper or glueware identified and specified?

Non-functional requirements

- Are there some non-functional requirements of specific importance in company-collaboration: integrability, testability?
- How are the non-functional requirements communicated to affected parties? How affected parties are identified?
- How fulfillment of non-functional requirements is verified from parts developed by the different parties?

3.2 Requirements management

- Can consistency between requirements and further work products be tracked during the development (when development is done externally)? Is there a need for that?
- Is there a need for uniform requirements management practices between collaboration parties (requirement ID's, traceability, change control)?
- When are RM practices planned? Is there need for RM plan?
- Does the used requirements management tool support company collaboration? What kind of support is needed?

Requirements documentation

- Are there criteria or requirements for how requirements should be documented (format, content, traceability)?
- How can it be ensured, that requirements documentation (definitions) is understood at the same way by each partner (training, terminology, workshops, communication, etc.)?

Requirements change management

- Is the number of changed requirements similar to in-house projects?
- How do different collaboration modes affect change management in general? (Phases: How are change requests done and delivered? How is the evaluation of changes done (impacts to other parts of software etc., costs, schedule, resources…)? How is the decision made to implement the change or not? How is the notification of accepted or rejected changes done? Implement: How is the change closed?
- How do different collaboration modes affect change decision making (authority, etc.)?
- How does different collaboration modes affect change impact analysis?
- How are changes communicated?

- How can it be ensured that the latest updated requirements documentation is used by each partner?

3.3 Architecture design process

- Should the architecture take into account the collaboration mode?
- Who is responsible for the architecture as a whole? (specially in joint development)
- Are architecture trade studies done?
- Do you determine realistic, independent estimates based on architecture? (for COTS, in-house, 3rd party development)
- Does buying components affect architecture?
- How can architecture design support different collaboration modes, partnering, etc. (be flexible, adaptable to anticipate changes in acquired parts)?
- Are licensed parts (COTS, OS) possible effects considered on high level architecture?
- Are licensed components integration issues analysed?
- Is glueware defined for selected licensed components?
- How is product line architecture opened for suppliers?
- How do you assure the security of product line architecture in collaboration?
- What is the meaning of system and SW architecture in practice?
- What kind of methods, techniques, tools or frameworks do you and your partners use in architecture design, evaluation and validation work?
- What kinds of differences and/or difficulties are found when general purpose architecture design and evaluation methods are used in collaboration?
- How do different collaboration modes affect to system and application architecture?
- What kinds of architectural viewpoints do you and your partners use and need in design activities?
- What kind of rationales do you and your partners attach to architecture description?
- What are the conditions and constraints for creating and designing architecture in large SW projects?

3.4 Software design process

- What are the differences between different collaboration modes vs. in-house development?
- Are there requirements for used design technique and/or tool?

3.5 Software construction process

- Is there anything different in different collaboration modes vs. in-house development?
- How collaboration affects in-house coding (schedule, tailoring, integration, maintenance, experience of employees)?
- Are there requirements for used coding language and/or tool?

3.6 Integration process

- How are dependencies to other suppliers taken into account?
- Do you have integration plan? How are collaboration modes affecting it?
- Are there specific tools that support integration in collaboration?
- Are responsibilities and authority defined and clear?
- Who performs the integration; are representatives from all parties present? Should they be?
- How is integration environment developed (in house, developed together with collaboration parties)?

3.7 Testing process

- Are there specific tools that support testing in collaboration?
- What tests are performed and by whom?
- Are responsibilities, information sharing (confidentiality issues) defined and clear?
- How is acceptance criteria defined?
- Are test cases shared?
- Who plans test cases and in which development phase(s)? (e.g., should test cases planned while developing requirements?)
- Are test environment shared?

- How are test results communicated (to the other collaboration parties)?
- How is defect management arranged (impact analysis, prioritization, communication)?
- Does the verification process differ between collaboration and in-house project?
  - Does implementation (by each sub-system) fulfill all specified requirements? (Verifying with customers, suppliers, other stakeholders etc.)
- Does the validating process (the system requirements against raw requirements) differ between collaboration and in-house projects? (How can it be ensured that raw requirements are understood the right way?)

3.8 Maintenance process

- Does the maintenance process differ while product has been produced by collaboration project (warranty, upgrading, resourcing…)?
- Should maintenance activities be recognized and considered in the planning phase of the collaboration project, or is there some critical phases when maintenance requirements should be checked during the project life-cycle?
- How are the responsibilities defined?
- Does a mitigation/alternative plan exist in case of problems?
- What kinds of mechanisms are used in case there are operational faults?
- How is the maintenance defined and agreed in contracts?

4. Support practices

4.1 Configuration management

- What kind of configuration management practices or tools should be agreed at the beginning phase of the collaboration project (e.g., common tools, databases, licenses, what details should be included also in contracts etc.)?
- When should CM practices be agreed (contract negotiations / project planning)? Is there need for documented CM plan?
- How can it be ensured that, e.g., the right version of the design or requirements documents, are used by each partner?

- How are sub-contracted parts treated in customer's CM process and tool (identification, traceability, change management)? Are they treated differently than artifacts produced by in-house development?
- If different configuration management tools or practices between customer and supplier are used, does the situation cause problems? If yes, describe problems.
- How are the contents of releases defined? What kind of information and documents each deliverable (e.g. SW sub-system package from sub-contractor) should contain? Are there any differences if product is developed by in-house or collaboration process?

4.2 Change management process

- How do problem reporting and handling process differ in collaboration vs. in-house development?
- How are change requests and changes communicated?
- How are decisions made (authority, who has a say, etc.)?
- How are change impacts analysed?
- Does the impact analysis of internal changes take into account the effects to other requirements and collaboration parties work?

4.3 Quality assurance

- How is quality assurance process implemented in collaboration?
  - How does the process differ from one implemented in the in-house development? (e.g., content of project plan, steering group meetings, members, acceptance process, etc.)
  - Does each partner have separate quality assurance processes?
  - How can a customer ensure that a supplier uses appropriate quality assurance process?
  - What quality assurance issues need to be agreed and planned in collaboration?
- How (and why) does the reviewing process differ between collaboration project and in-house project (e.g., in-house reviews, acceptance reviews)?
  - Are there separate review processes and practices applied by each partner?

  - Which reviews should be held together (e.g. acceptance reviews)? Who is responsible to arrange them?
  - How and when are decisions made about participants of review sessions from each partner (planning phase, during some development phase)?
  - Is review criteria specified (/checklist) for acceptance?
  - How is it reviewed that supplier follows necessary standards (telecommunications etc.)?
  - Do you identify high risk areas of supplied software?
  - Are reviews focused based on risk analysis?
  - Are end users (operators etc.) included to technical reviews of the supplier produced code?
- What work products should be reviewed in which level / by each partner / with both partners? Member of steering group, members of review group and how those groups differ between in-house and collaboration projects?

4.4 Documentation

- How (and why) do the documentation management practices differ between partners?
  - Which documents will be shared between customer and supplier, and how (e.g., documentation management system, common or distributed (replicated) databases (/areas), licenses, access to customer's information system?
  - Are there needs to develop new document templates (or update current templates with new parts or items) in collaboration project?
- How do documents (templates) differ from the ones produced either in collaboration or in-house developing? For example,
  - Contract: What should be agreed; e.g., deliverables of each partners, ownership and rights of use, NDA information, security policies in general, documentation formats for deliverables (e.g. editable formats like ".doc" or ".xls" or view only like "html" or "pdf")?
  - Project plan: Detailed planning includes (deliverables by partners and detailed timetable during life-cycle of developing), review and acceptance process, change request and handling process, etc.
  - Release notes (e.g., changes done, features added, template for notes): By whom content and issues of release notes should be de-

termined? What kind of information is important from viewpoint of customer (tracking, change handling…)?

- Other document templates? Which ones?

4.5 Improvement process

- Are there shared processes with parties?
- Is there shared process improvement work?
- Is effectiveness of collaboration evaluated on a regular basis?
- Are there dependencies between the partners' processes?
- Are there new processes needed in different collaboration modes? Are they identified and described? Are they assessed on regular basis? Are improvement actions taken accordingly?

4.6 Human resource management process

- Are new roles and skills required because of company collaboration?
- Are subcontractors trained?
- Do COTS providers provide training?
- Is any other special training arranged or needed because of collaboration (co-operation skills, leadership skills, collaboration tools)?
- How can effective interaction between individuals and groups be supported?
- Is group and individual performance monitored to provide performance feedback and to enhance performance?

4.7 Infrastructure process

- Are there specific tools that support collaboration (development and information management/exchange tools (and technologies))?
- What requirements do different collaboration modes bring to infrastructure?
- Are there any means of control for other parties infrastructures (tools used, etc.)? Is it needed?

4.8 Co-operative work practices

- What kind of methods, techniques and tools exist for co-operative work?

- How mature is the organisation towards co-operative work (coupling of work, collaboration readiness, technology readiness)?
- Are the costs and benefits of co-operative work analysed?
- How are the co-operative work teams organised?
- How are the communications arranged in co-operative work?
- How does the distance and cultural issues affect the co-operative work?

5. Other

- Anything else important related company collaboration?
- If you could change one thing in company collaboration, what would it be?
- What do you see as the strengths in company collaboration in your organisation?

Appendix B: GSE questionnaire

Background information

This questionnaire is part of industrial inventory carried out in the MERLIN project in order to gather existing experiences, problems and solutions of company collaboration. Answers are treated confidentially and will be used as input for industrial inventory summary report of the MERLIN project. Before sharing the results with the MERLIN project consortium in the summary report, you'll have opportunity to review the report.

Organisation:	
Name:	
Position/role (describe your responsibilities briefly):	
Phone:	
E-mail:	

General collaboration practices

What kind of collaboration modes does your organisation use? - Are there differences in importance and effect in different collaboration modes?				
Collaboration modes (one viewpoint based on organizational interdependency):	Lots of	Some	None	Comments:
- Joint R&D (or partnering):				
o Joint ventures. Joint ventures are organisational units created and controlled by two or more parent-companies.				
o Joint development agreements. Joint development agreements cover technology and R&D sharing between two or more companies in combination with joint research or joint development project.				
- Customer - Supplier relationships. In customer-supplier relationships, customer is the organisation that is buying the software work (and technology and knowledge) from the supplier. Work may be based on requirements given, or modification of existing COTS or open source code. Customer may also hire workers from supplier in so called body-shopping. Supplier is the organization that provides the software work to the customer. We have identified three main types of relationships:				

o Requirements based subcontracting				
o Body-shopping				
o MOTS, (COTS), (open source)				
- Technology exchange/Licensing. By technology exchange/licensing, company is granted the right to use a specific patented technology in return for a payment. Companies may also define open interfaces to products that allow any interesting party to create software/services to the product. Types of technology exchange/licensing include:				
o COTS				
o Open source				
o Open architectures				
Other, what?				

Above categorization is one view to company collaboration, other view is to identify collaboration from equity or non-equity perspective. Collaboration can also be horizontal or vertical, indicating relative positions of the participating companies on the value-chain. In practice, however, product development is typically a mixture of the above, but when analyzing the effects of the collaboration, the above categorization is useful, as it provides different viewpoints to support the analysis. Please, write your further comments here:
What is the general motivation for collaboration? (E.g., to acquire expertise, to save money, to save time, not to invest to non-core competence areas, to share risks)
What are the main risks and problems in collaboration?
Who, how and in what level selects the collaboration mode? (E.g., project manager for a project according to a defined process, product manager for the product-line, steering group for the organization, ad-hoc case by case etc.)
Do you have specific roles and responsibilities for collaborative development? What, please describe responsibilities? (For example, collaboration manager, sub-contract managers, product integrator)
Are communication practices and subjects (e.g., change management) defined in collaborative development? What are the subjects?
At which points of R & D are collaboration issues taken into account (project planning, requirements specification, architecture analysis, etc.)?

Is company collaboration taken into account in your organisation's internal processes? How?
Are the process interfaces defined with collaboration parties in co-operation?
Are there specific tools used for collaborative development?
Would you need specific tools for collaborative development? For what purposes?

Critical points in collaboration

Please consider your company's experiences relating to collaboration in SW development and rate the criticality of the following topics. See descriptions of the topics from enclosure 1.

Which are the "critical points" when dealing with collaboration?	Not at all critical	Not critical	Critical	Very critical	Comments / reasoning for rating (comment also, if rating is specific for certain collaboration mode)
Management practices					
- Collaboration strategy					
- Contracts					
- Project management					
- Risk management					
- Collaboration management					
- Change management					
- Quality management.					
Engineering practices					
- Requirements development					
- Requirements management					
- Architecture design					
- Software design					
- Software implementation					

- Integration					
- Testing					
- Maintenance					
Support practices					
- Configuration management					
- Quality assurance					
- Documentation					
- Improvement process					
- Human resource management					
- Infrastructure					
- Co-operative work					

Thank you very much for completing this questionnaire!

All answers will be handled confidentially.

Appendix C: Papers II and VIII, are not included in the PDF version. Please order the printed version to get the complete publication (http://www.vtt.fi/publications/index.jsp).

PAPER I

Collaborative embedded systems development

Survey of state of the practice

In: Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), March 27–30, 2006, Potsdam, Germany.

Collaborative Embedded Systems Development: Survey of State of the Practice

Jarkko Hyysalo
University of Oulu
Jarkko.Hyysalo@oulu.fi

Päivi Parviainen
VTT Technical Research
Centre of Finland
Paivi.Parviainen@vtt.fi

Maarit Tihinen
VTT Technical Research
Centre of Finland
Maarit.Tihinen@vtt.fi

Abstract

This paper describes the results of a survey about the problems of and solutions for collaborative SW development. The survey was done through several interviews of companies doing collaborative development and also through a literature search to find already published experiences and solutions. As a result, we found that the literature focuses on solutions for more general issues like communication and team building, and industrial problems are related to specific engineering tasks. Mapping and practical examples of general solutions to specific tasks are needed to support collaborative software development.

1. Introduction

The development of embedded systems is a multidisciplinary and highly complicated process with tight time-to-market requirements. Thus, nowadays, embedded products are typically not developed by a single company alone, but instead globally in collaboration between subcontractors, third party suppliers and in-house developers.

This kind of collaborative product development offers many opportunities, such as potential savings in development times and costs, and being close to several customers. However, collaborative development is at the same time highly challenging; for example, the development teams are usually dispersed and this alone places high demands on communication, teamwork and working methods. In addition, different time zones and distances make communication more difficult than in local development. Thus, the whole development process, in general, differs significantly from local (single-site) development process.

The purpose of this paper is to describe the state of the practice in collaborative embedded systems development from a multi-company viewpoint, though applicable single-company experiences are also included. The main goal of our work was to gain a view of current collaborative practices; what the most problematic or critical issues relating to collaborative work are and what the most important areas are that should be the focus of research activities. The work was done as part of a large research project, called Merlin[1], in the year 2005. Merlin is a three year ITEA project, comprising of industrial and research partners from three countries. The main aim of the project is to support effective collaboration between companies via developing and tailoring supporting technologies and processes. This paper presents these findings.

The majority of existing publications focus on distributed development within a single company, i.e., multi-site development. See for example, Motorola [1], Alcatel [2], and Lucent Technologies [3, 4]. Some studies focus on the effect of or solutions provided by distributed development for a specific issue, such as requirements engineering [5], software configuration management tool support [6], or process support system [7]. A more detailed discussion on the published experiences and solutions is to be found in sections 3 and 4.

1.1. Background

In this paper, collaboration refers to product development activities that involve two or more companies, departments or customers combining their competencies and technologies to create new shared value while, at the same time, managing their respective costs and risks. The entities can combine in any one of the several different business relationships and for highly different periods of time, ranging from short periods needed for exploiting a particular innovation or business opportunity, to long-term on-going relationships. (Adapted from [8].)

There is no established definition for collaboration modes. Collaboration modes, as used in our survey, (adapted from [9]) include the following:

- **Customer - Supplier relationships.** In customer-supplier relationships, customer is the organisation that is buying the software work (and technology and knowledge) from the supplier. The work may be based on specified requirements, or modification of existing COTS or open source code. The

[1] The Merlin project: Embedded Systems Engineering in Collaboration, 2004-2007.
URL: http//www.merlinproject.org/

Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)
0-7695-2546-6/06 $20.00 © 2006 IEEE

customer may also hire workers from the supplier, which is referred to as "body-shopping". Supplier is the organisation that provides the software work to the customer. Three main types of relationships are identified:
  - Product structure based subcontracting.
  - Body-shopping.
  - MOTS (Modified-Off-The-Shelf).
- **Technology exchange/Licensing.** By technology exchange/licensing, a company is granted the right to use a specific patented technology in return for a payment [9]. Companies may also define open interfaces for products that allow any interested party to create software/services for the product. The types of technology exchange/licensing include:
  - COTS
  - Open source
  - Open architectures
- **Joint R&D (or partnering)**
  - Joint ventures. Joint ventures are organisational units created and controlled by two or more parent-companies. [10]
  - Joint development agreements. Joint development agreements cover technology and R&D sharing between two or more companies in combination with a joint research or joint development project. [11]

The above categorisation is one way of viewing collaboration; another would be to identify collaboration from an "equity or non-equity" perspective. Collaboration can also be horizontal or vertical, indicating the relative positions of the participating companies on the value chain. While, in practice, product development is typically a mixture of the above, when analysing the effects of the collaboration, the above categorisation is useful as it provides different viewpoints to support the analysis.

In the following sections, we first explore the findings of our industrial survey and the experiences gathered from literature to gain an insight into the state of the practice in collaborative embedded systems development. The suggested solutions are mapped to related problem issues to find out what areas are covered and what the needs are for further improvement. That is also our main research question: What are the areas that need improvement in collaborative development in the embedded systems domain?

1.2. Inventory scope

The industrial part of the survey was carried out by performing interviews and studying the existing material of the companies participating in the Merlin project, including process descriptions, templates, and guidelines. The interviews were carried out using a specific framework. A total of 6 companies participated in the inventory and 12 interviews of senior managers, project managers, software developers and testers were carried out. The industrial partners represent several divergent embedded SW business areas: mobile and wireless systems, data management solutions, telecommunications, IT services, and consumer electronics.

The addressed topics are introduced on a general level in Table 1.

Table 1. Inventory topics

Management practices	Collaboration strategy
	Contracts
	Project management
	Risk management
	Collaboration management
	Quality management
Engineering practices	Requirements development
	Requirements management
	Architecture design
	Software design
	Software implementation
	Integration
	Testing
	Maintenance
Support practices	Configuration management
	Change management
	Quality assurance
	Documentation
	Improvement process
	Human resource management
	Infrastructure
	Co-operative work

In addition to the industrial survey at Merlin partners, literature was also searched for published experiences and solutions regarding collaboration. Literature was surveyed for subjects ranging from development issues and success factors to tool support in collaboration. The experiences found from literature are mostly from distributed development within a single company, but some multi-company efforts are also reported [e.g., 12, 13, 14, 15, 16, 17].

2. General findings

In this section, the general findings of the industrial survey based on the interviews at the Merlin companies are presented and discussed.

2.2. Collaboration modes

The most common collaboration mode was product structure based subcontracting (the organisation was split

Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)
0-7695-2546-6/06 $20.00 © 2006 IEEE

into sites along the lines derived from product requirements or architecture [18]). The companies also showed a lot of body-shopping and distributed (or multi-site) development.

Joint ventures were not a common practice, none of the companies were using them. Joint development agreements with other companies were a lot more common, especially the bigger companies were using this strategy to some extent or to a great extent. The use of COTS was also relatively common, though there was a lot of variation between the different companies: while SME's seemed to use COTS to a lesser extent, bigger companies turned out to make more use of them. However, also SME's were expected to increase their use of COTS in the future.

The use of open source software also varied a lot between the different companies; some didn't use them at all, whereas some made extensive use of them. The reasons for not using open source software included, for example, a principle that all that's in the product should be owned by the company itself; licensing and public releasing are likely to reduce competitive advantage and also problems in responsibilities.

The following collaboration activities were also identified among the Merlin partners:

- Co-operation with experts of the domain (not software experts).
- Joined company with two other companies that, for example, offers a shared resource pool for subcontracting, hold shared trainings and support each other in specialised expertise areas.
- Participation in domain specific forums, e.g., to influence standards.
- Outsourcing of maintenance.
- Subcontracting hardware development to third parties.

The decision to choose this collaboration mode was mostly done case-by-case and no clear rules or guidelines seemed to exist in any of the companies. Subcontracting was also most often done with proven subcontractors.

2.2. Motivation and risks

The three most commonly mentioned reasons for collaboration were:

1. To reduce development costs.
2. To acquire competence (technology competence or knowledge of a specific market).
3. To avoid investing in company's non-core competence areas.

Further reasons included potential timesaving, establishment of new business opportunities with new partners, flexibility with respect to the number of in-house resources, and availability of in-house resources. In some cases, e.g. regarding COTS developers or consulting companies, the whole business is collaborative.

The general risks mentioned had to do with the openness of communication between partners; for example, problem hiding may be a problem in customer-supplier relations. Further, unclear assignments, trust between partners, agreeing on intellectual property rights and the reliability of the partners' development schedule were seen as risk factors for any mode of collaboration. From the supplier's or licensor's viewpoint, the risks mentioned concerned the continuation of the collaboration in the future and predicting the most sales-boosting features right during roadmapping. On the other hand, from the customer's point of view, the quality of the acquired product (e.g., reliability and performance) and becoming too dependent on one partner were seen as risks. Finally, competence issues, such as competence of new partners, and weakening of one's own competence were also mentioned as risks.

2.3. Success factors

The success factors for collaboration and co-operation were also discussed with the interviewees. The following success factors for collaboration were mentioned

- Fluency of co-operation, e.g., a face-to-face kick-off meeting helps in establishing this.
- Good understanding between partners of each other's work.
- Mutual benefit from collaboration, i.e., real need for co-operation.
- Partners complementing each others' expertise when, for example, it has been easy to reach an agreement on work distribution and decision authorities.

2.4. Collaboration tools

In most cases no specific tools were used for collaborative development. However, same tools for specific activities were often used between partners. These included tools for defect management, configuration management, requirements management and change management. This would often mean that the customer's (or integrator's) tools were used, but not in every case, as sometimes the subcontractor might be so influential or significant, that the customer just could not think of demanding the use of any specific tools. Some work may also require using highly specialised tools, e.g., in integration testing, so that the work needs to be done with customer/integrator tools and sometimes even at the customer's/integrator's premises.

Some companies also use a shared intranet area in collaborative work. On the other hand, some other companies do not recommend allowing outside companies to access the intranet.

Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)
0-7695-2546-6/06 $20.00 © 2006 IEEE

For communication, e-mail and phone were the most commonly used tools.

Most tools were reported not to support collaborative development well enough. Especially the need for better change management tools was often mentioned. Regardless of the fact that specific collaboration tools were not used much, there was an apparent need for awareness tools, especially in managing the transition between synchronous and asynchronous work [19].

3. Critical points and problems

In this section, the critical points and problems presented in literature and determined through the industrial inventory are discussed.

3.1. Industrial inventory

One important viewpoint in the interviews was to gain knowledge of the most critical points in collaboration. According to the interviews, the most important/complex points were:

- road-mapping
- contracting and requirements definition
- project planning
- architecture analysis/design
- integration
- change management
- co-operative work

The list shows that the industry focuses on problems from a technical view of point. Each of the listed points is discussed in more detail below, including solutions when available.

Roadmapping: Taking into account and prioritising the customers' future requests or suppliers' future directions is a complex task, especially for several customers with varying requests.

Contracting and requirements definition: The more detailed the prepared specification of the work is, the better (within a reasonable degree of effort). Thus, all collaboration partners have the same view/shared understanding of what is to be done and if that is written down well, fewer conflicts will occur.

Project planning: It is important to clearly define status reporting practices and change management procedures, including the details on reporting channels, decision authorities and escalation channels. Further important issues in collaborative project planning are the identification of dependencies between partners - e.g., the interdependencies of the subsystem deliveries - and taking these into account in project schedules.

Architecture analysis/design: Architecture is one of the key technologies enabling successful collaboration. In particular, integrability is a key issue. The responsibilities and authorities should also be explicitly assigned regarding architectural analysis, design and changes. Ensuring that all partners understand the architecture correctly is difficult and the required level of communication is often underestimated.

Integration phase (testing): While integration is often the most time- and effort-consuming activity even in in-house product development, collaborative product development brings additional complexity, e.g., new actors, and communication requirements. An efficient integration strategy should be defined and communicated to everyone. Establishing an integration culture (source code is only finished when it has been integrated) was also recommended. Continuous integration should be pursued whenever possible and big bang integration avoided as continuous integration is likely to reveal problems earlier. In addition, allocating a sufficient amount of time and capacity for integration is of great importance. On the other hand, the availability of the same test environment among the partners and the availability of required expertise may be an issue.

Change management: The change management process should be adjusted to the collaborative development environment (e.g., proposals, scope of impact assessment, communication, viewpoints that need to be taken into account in decision making).

Co-operative work: No specific methods, techniques or tools are used for co-operative work. E-mail and phone discussion were the most commonly used practices. However, communication was seen crucial for the success of the project. Openness of communication and multi-site/multi-partner culture were considered very important for collaboration success.

The following causes of problems in co-operation were mentioned: problems with time difference, cultural differences, and knowledge of the product not being at a high enough level. Ensuring a common understanding was also found a complicated issue.

The working practices deemed useful based on the multi-site development experiences include face-to-face meetings at the beginning of the project and regular meetings during the project between partners (architects, configuration managers, (sub)project leaders).

3.2. Problems derived from literature

The reported literature findings are categorised according to Carmel's [12] classification:

- loss of communication richness
- coordination breakdown
- geographical dispersion
- loss of "teamness"
- cultural differences

In addition to the above, we also found some topics that did not fit into any of Carmel's categories. The Carmel classification was selected as the reported problems were

quite universal and they were closely linked to co-operative work. The literature approaches were not as technical-centred as industrial inventory interviews. In the following, the listed points are discussed in more detail.

Loss of communication richness: Shared artefacts (e.g., flip charts, whiteboards, tack boards, and walls) can normally be used to show the work in progress or as reference material [20]. In addition, distribution may hinder informal or unplanned communication [3]. In distributed development, all this has to be managed and supported through groupware or the like, or else the richness of communication will suffer. A loss of communication richness makes it necessary to concentrate on the quality of documentation; poor and inadequate documentation is likely to cause inefficiency in collaborative development [14].

Coordination breakdown: Different practices and processes of teams cause problems while working in distributed settings. The need of specific practices and processes for collaboration was also often underestimated. Collaboration also sets additional requirements for planning; for example, the needs for coordination between teams and the procedures for how to work with partners were often not paid enough attention to in planning collaborative projects [5, 15, 22]. Difficulties in identifying distant colleagues and problems with not knowing whom to contact across sites were also reported [22]. Battin et al. [1] reported a number of software integration problems which were due to a large number of independent teams. The authors also stated that software configuration management is a challenging task in globally distributed development.

Geographical dispersion: Overall, communication is difficult in geographically distributed development, for example, if there is a lack of overlapping working hours due the time zone differences [5]. Distribution also causes challenges to requirements development [5, 22]. For example, it's more difficult to establish common understanding when different stakeholder groups specify requirements across distances. Dividing the tasks and work across development sites is also difficult due to the restrains of the available resources, the level of expertise, and the infrastructure, for example [14].

There can also be great differences in governance, which makes it more difficult to manage inter-site work dependencies and to coordinate and control the distributed work [8, 1]. Furthermore, lack of vendor support is a problem in distributed development as some third-party tools do not have worldwide support [1].

Loss of "teamness": The role of trust is always significant in collaboration and contracting, because it is very difficult, if not impossible, to make perfect contracts. Lack of informal communication has negative impact on knowledge management and other issues, such as trust [5, 22]. In addition, different processes and practices tend to diminish teamness [1].

Cultural differences: If the distribution crosses cultural boundaries, the implications for legal issues, knowledge-transfer, development and project management and quality management may be amplified, while language, time, and infrastructure issues can make the process even more challenging. [5, 23]

There are differences in tacit knowledge consisting often of habits and culture that we do not recognise in us. By definition, tacit knowledge is not easily shared, causing, for example, reluctance to use international developers and resistance towards global software development [1, 14].

Other problems: Uncertain requirements and implementation technologies were also deemed problematic; for example, lack of clarity about who is a stakeholder, disparity of power and/or resources among stakeholders, and complex problems that are not well defined. [16]

Collaborating companies are typically operating in a nexus of contracts. The business relation between the parties consists of the trading relation made up by numerous interactions, some of which may involve contracts, but often will consist of enquiries, discussions of plans, and sorting out problems [24]. From a contractual point of view, there are several types of contracts that are granting rights or creating liabilities. It is essential for a company to be aware of all these rights and liabilities and a mixture of them.

The roles and responsibilities of different stakeholders should be clearly defined. This takes a lot of time and requires various kind of expertise from the company. [25]

Inadequate tools, i.e., which are not synchronised seamlessly, not efficient enough, or not spontaneous enough to support informal communication, were often reported as a problem. Other problems related to tools are: it is difficult to access databases, the quality is not high enough, data formats may vary, and different teams may have different versions of the same tool. [2, 26, 21, 27, 3, 14]

3.3. Discussion

In this section, the results of industrial and literature inventory - congruent and divergent findings - are discussed together. The findings are discussed from the viewpoint provided by the Carmel categories.

Loss of communication richness is related to physical distance and time zone differences. Both literature and industrial interviews named distance as an issue. Physical distance hinders face-to-face communication effectively and creates other obstacles, such as complicated use of shared artefacts, and lessening of the amount of informal communication. The loss of communication richness also creates miscommunication. In addition to these issues, different views of remote teams cause real, intangible problems to development. These are not solved as easily in distributed settings as in collocated development because of all the communications related issues. For example, it is

difficult to bring about synchronous communication and this creates obstacles in resolving miscommunication, misunderstandings and small problems as they cannot be resolved asynchronously as easily as in synchronous communication [32].

There are differences between our findings and the problem issues referred to in literature and industrial experiences. In the surveyed industrial experiences no references were made to the lack of informal or unplanned communication or poor documentation, and literature did not name miscommunication or different views as problems. This indicates that different projects face different problems and not all of the listed problems are issues in all of the projects.

Coordination breakdown revealed many similar problem issues in literature and in industrial settings. Team coordination is clearly an issue as are also integration related problems. Project planning seems to be a significant problem, which is also reflected in an underestimated need of collaboration practices and processes.

The differences between literature findings and our industrial experiences are noticeable. Literature suggests that different practices and processes used by different teams complicate the coordination efforts and, if there is a need to find contact persons across sites, it may be difficult as it is not clear who is responsible for what. Difficulties in configuration management are a further hindrance to coordination efforts. On the other hand, our industrial experiences brought up some work monitoring related problems.

Geographical dispersion did not reveal similar problems in literature and in industrial experiences. Literature discussed the difficulties of work and task allocation across sites and also the delay caused by the distributedness of the project. Distance complicates the arrangement of face-to-face meetings and they cannot be arranged as often as they are needed. Furthermore, great time zone differences create a lack of overlapping working hours, which in turn hinders, for example, communication and exchange of shared artefacts. Although, in theory, this could be used as an advantage to enable work around the clock, literature did not mention any successful experiences on this.

Industrial interviews revealed problems related to the availability of resources. It was mentioned that it felt easier when the same people were working together through the whole project, but when many sites were participating in a project it was likely that some changes in personnel would happen.

Loss of "teamness", both literature and industrial experiences found trust to be an important factor. Lack of trust is clearly an issue that calls for action; however as creating relationships and trust takes time, it is often difficult to tackle this problem.

Cultural differences related issues could be found both in literature and in industrial experiences. Requirements specification over distance was already mentioned, but across cultural boundaries this task becomes even more demanding. This is also related to differences in tacit knowledge and cultural boundaries in general (language etc.). Literature also discussed the reluctance of using international developers, which was, however, not revealed in our interviews.

A number of **further problems** were revealed by the review of literature and industrial experiences. These are uncertain requirements and implementation technologies, the high number of different contracts and interactions, insufficient definition of roles and responsibilities, different tools or versions, inadequate tools, synchronised problems and problems with data security and access to databases or another organisation's resources etc. Besides these, the interviews revealed further issues, such as reluctance to changes (as changes usually cost money), and lack of commitment. As not every organisation or team is fully committed to the project, this may be reflected in the development and also in the end product. Furthermore, since architecture related problems were one of the major issues according to our interviews, they should be taken into serious consideration when working in distributed settings.

4. Potential solutions

There are plenty of solutions for collaborative work in the literature. In this section, the solutions are presented according to industrial inventory topics (Table 1):

- Management practices
  - *IPR management*
  - *Development strategies*
  - *Synchronisation of main milestones*
  - *Clear decision-making practices*
  - *Decoupling the work across different sites*
  - *One project leader*
  - *Fully accountable teams*
  - *Relationship management*
  - *Informing and monitoring practices*
- Engineering practices
  - *Architectural principles, "architecture lite"*
  - *Frequent deliveries*
  - *Several iteration cycles*
  - *Frequent and incremental integration*
  - *Up-to-date documentation*
- Support practices
  - *Multi-site configuration management*
  - *Interactive process*
  - *Sufficient communication means*
  - *Proper network-infrastructure*
  - *Establishment of peer-to-peer links*
  - *Cultural awareness*

Management *practices:* Contracts and other legal aspects, e.g., contract management, legislation and legality

were considered important, while *IPR management* was referred to as the primary tool for managing these issues.

Project management practices are in a central role while solving problems in collaborative work. *Development strategies* should be planned properly with visible goals [31]. It should be defined at the beginning which teams are involved and what they will do in each location [2]. *Synchronisation of main milestones* between partners/sites with clear entry and exit criteria is one of the practices facilitating collaboration [17]. At project start, the respective project targets, such as quality, milestones, content, and resource allocation should be agreed on and communicated to the relevant parties [2]. In order to enhance the *decision-making practices* for collaboration management, defining the correct forums for different types of decisions, informing stakeholders about decisions and arranging network level steering group meetings are suggested [4]. In addition, project level coordination should be used as a supporting structure, for example, by creating a project level steering group with members from all organisations and sites [4]. There should also be inter-organisational groups with weekly meetings (project level, team level, etc.) via appropriate means (e.g. by teleconferencing). During meetings, problem solving and decision-making should be facilitated as they provide transparency and facilitate later electronic communication [4]. It is also useful to ensure that commitments exist in written and controlled form [2].

Herbsleb & Mockus [23] recommend *decoupling the work across different sites* so that these can work as independently as possible. The authors suggest different ways to do this, for example, by process steps or dividing the product according to its structure and developing different parts in different places, which results in an organisational structure that follows the product structure.

Ebert & De Neve [2] suggest having *one project leader* or responsible for achieving project targets and assigning this person a project management team representing the major cultures within the project. It is further suggested that the *teams* should be *fully accountable* and responsible for their results. Reifer [32] also acknowledges the importance of *relationship management* and proposes for projects to assign a dedicated relationship manager for key suppliers. Several relationship building practices are suggested: All communication, especially face-to-face meetings help to build a good cooperative relationship. Face-to-face contacts are needed also with distant sites to give "faces" to distant sites. Further, organisational charts help to recognise persons. A common kick-off meeting is a good start. Other recommended practices are, for example, circulating meetings or trainings, and planning or problem solving meetings. [18, 4]

Informing and monitoring practices comprise a number of different practices, such as weekly meetings, progress reports, and a travelling steering group. Informing and monitoring should be followed-up in all directions [18]. The parties should comment all the points in the follow-up reports that include tasks done, open questions, problems, and future outlook. Weekly meetings inside a subgroup are also proposed [4]. The key risk factors should be identified and an approach should be developed to mitigate them, while continuous monitoring of the key risk factors is also recommended [1, 2].

Engineering practices: *Architectural principles* as a solution for architectural issues and actively working towards "*architecture lite*" are proposed as useful practices [1, 33]. Architecture lite refers to low coupling among network elements, well defined interfaces, and concise semantics for the network element behaviours. This kind of architecture describes the system architecture at a high level of abstraction, including both static and dynamic views. System metaphor can be used to guide the development.

Frequent deliveries of code, and *several iteration cycles* and builds help in creating transparency for the process. In addition, *frequent and incremental integration* and testing are suitable for distributed use. Early checking ensures that all parties have understood their tasks correctly. This is especially useful during the software development phase [18, 1]. All *documentation* is also recommended to be revised and *updated* to reflect the current state of the development [17].

Support practices: Herbsleb and Moitra [17] point that *configuration management* involving transmission of critical data and multisite production must be well planned and executed in collaborative work. A common software configuration management tool with multisite replication and a centralised bug repository in useful in distributed development [1].

Collaborating companies do not have to switch over to a single common process if they have good processes of their own, but instead they should focus on synchronising the main milestones, and use iteration cycles of similar length and frequent builds. Common milestones and work products synchronise communication, also facilitating both follow-up and communication. The collaborating companies being able to use their own development processes provides a faster start for a project, while also making it easier for several companies to collaborate [18, 4, 1]. The decision about using separate processes should be made at the beginning of the collaboration in order to benefit the whole development process. An *interactive process model* based on accepted best practices that allows tailoring the development process for the specific needs of a project or even a team was reported to be useful [2].

It is obvious that *sufficient communication means* should be provided [2] i.e., proper tools and support for them. Furthermore, it is suggested to integrate the heterogeneous tools for system development, not only for abstract concerns (e.g., analysis), but also for concrete activities (e.g., programming, unit testing, conducting workshops), and to provide support for the relationships between activities and concerns [30]. International support contracts are needed in order to have reliable support for tools from

third-party vendors. In addition, centralised bug reporting system for bugs found in (third-party) tools and products improves the vendor's responsiveness, as it alleviates version management and overcomes the lack of local vendor support. [1]

Proper network-infrastructure enables fast data/information exchange between sites. A replication of project repository is often needed. Furthermore, distributed working requires establishing common rules and procedures as well as problem solving practices (e.g., how to use configuration management systems and how to escalate problems). Furthermore, setting up a project homepage summarising project contents, progress metrics, planning information and team specific information is suggested [2].

Strategies like communicating internally, maintaining contact with all participants help in building trust [31]. *Establishing peer-to-peer links* is suggested, denoting, for example, communication link persons or liaisons between companies established at all organisational levels, e.g., subcontracting managers, project managers, and also developers [18]. An organisation chart and roles help to find the correct person to contact. The roles should include communication requirements and identify which roles need to communicate with each other between companies. Other possible beneficial practices are setting up a mailbox for questions, chat between developers, discussion lists, and project wide mailing list with well-explained questions. These are useful practices especially in the implementation phase. [18, 4]

Also *cultural awareness* should be created involving all the cultures represented in the development teams. This can be done, for example, by circulating management across sites and cultures ("cultural liaison") or by setting up mixed teams of different cultures to create an awareness for cultural diversity. This will also give ideas for how to cope with the diversity along with creating team spirit. [2]

Finally, it should be noted that competence and experience in general are likely to improve the chances for successful collaboration [1].

5. Conclusions

Based on our survey, the most common collaboration mode was product structure based subcontracting while the main motivation for collaboration was most often to save money. However, acquiring competence not available in-house was another common motivation. While the areas seen critical in collaborative product development varied, those most commonly perceived as critical were contracting, change management, requirements development and requirements management. The areas most commonly seen as non-critical were software implementation within engineering practices and improvement process and human resource management within support practices.

This survey revealed that the approaches represented by literature, on one hand, and industrial practitioners, on the other, towards problems related to collaborative work are different. The industry emphasizes technical aspects and detailed problems concerning engineering practices, while the literature focuses on solutions for more general issues like communication and team building. Our survey establishes that there are a lot of solutions available especially for management and support practices. In the literature, then again, we could only find few solutions for engineering practices.

Based on the survey of the state of the practice, our future work will concentrate on defining practical solutions for the most critical areas and activities in collaborative embedded software development. We will make an attempt to apply the generic solutions found in literature to real life problems and report the experiences from these cases. For example, communication is considered to have quite a central role, so the communication needs and points have to be identified and concerned. This means that practical solutions should include, for example, a development process that describes the roles and responsibilities of the different parties, mapping these to each development phase and/or activity within the big picture of organisations developing software in collaboration.

6. Acknowledgements

This paper was written within the Merlin (http://www.merlinproject.org) project, which is an ITEA project, number 03010. The authors would like to thank the support of ITEA (http://www.itea-office.org/index.php) and Tekes (http://www.tekes.fi/eng/).

References

[1] Battin, R., Crocker, R., Kreidler, J. & Subramanian, K., 2001, Leveraging Resources in Global Software Development. In IEEE Software March/April 2001.

[2] Ebert, C. & De Neve, P., 2001, Surviving Global Software Development. In IEEE Software March/April 2001. pp. 62-69.

[3] Herbsleb, J., Mockus, A., Finholt, T. & Grinter, R., 2001, An Empirical Study of Global Software Development: Distance and Speed. In Proceedings of the International Conference on Software Engineering, 2001, Toronto, Canada, May 15-18. pp. 81-90.

[4] Karlsson, E-A., Andersson, L-G., and Leion, P. Daily build and feature development in large distributed projects. In the Proceedings of International Conference on Software Engineering (ICSE) 2000. ACM Press, Limerick, Ireland.

[5] Damian, D. & Zowghi, D., 2002, Requirements Engineering challenges in multi-site software development organizations. In Requirements Engineering Journal, 8, pp. 149-160, 2003. The paper is a revised version of the paper entitled "The impact of stakeholders' geographical distribution on managing

requirements in a multi-site organization" published in the Proceedings of the IEEE Int'l Conference on Requirements Engineering, 2002

[6] Surjaputra, R., Maheswari, P. A Distributed Software Project Management Tool. In IEEE Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, USA, 1999.

[7] Gianpalo, C., Ghezzi, C. Design and Implementation of PROSYT: A Distributed Process Support System. In IEEE Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, USA, 1999.

[8] Welborn, R. & Kasten, V., 2003, The Jericho Principle, How Companies Use Strategic Collaboration to Find New Sources of Value, John Wiley & Sons, Inc, Hoboken, New Jersey.

[9] Duysters G. and Hagedoorn J. A note on organizational modes of strategic technology partnering. Journal of Scientific and Industrial Research. Volume 58, August September 2000, pp. 640-649.

[10] Williamson, O.E, 1996. The mechanisms of governance. Oxford, Oxford University Press

[11] Hagedoorn, J., "Inter-firm R&D Partnerships—An Overview of Major Trends and Patterns since 1960," paper presented to the Workshop on Strategic Research Partnerships sponsored by the National Science Foundation and convened at SRI International, Washington, D.C., October 13, 2000.

[12] Carmel, E., 1999, Global Software Teams: Collaborating Across Borders and Time Zones, Prentice-Hall, Upper Saddle River, N.J.

[13] Paasivaara, M., 2003, Communication needs, practices, and supporting structures in global inter-organisational development. In ICSE International Workshop on Global Software Development, Portland, Oregon, 2003, IEEE Computer Society.

[14] Herbsleb, J. & Moitra, D., 2001, Global Software Development. In IEEE Software, March/April 2001. pp. 16-20.

[15] Paasivaara, M. & Lassenius, C., 2004, Collaboration Practices in Global Inter-organizational Software Development Projects. In Software Process Improvement and Practice, 2003; 8. pp. 183-199.

[16] Gray, B., 1989, Collaborating: Finding Common Ground for Multiparty Problems. San Francisco: Jossey-Bass.

[17] Gulati, R., 1998, Alliances and networks. Strategic Management Journal 19: 293-317.

[18] Grinter, R. E., Herbsleb J. D., & Perry, D. E., The Geography of Coordination: Dealing with Distance in R&D Work, In proceedings of the international ACM SIGGROUP conference on supporting group work, 1999, pages 306-315

[19] Wierba, E. Finholt, T. & Steves, M., 2002, Challenges to Collaborative Tool Adoption in a Manufacturing Engineering Setting: A Case Study. In Proceedings of the 38th Hawaii International Conference on System Sciences 2002 (HICSS-35'02).

[20] Olson, J., Covi, L., Rocco, E., Miller, W. & Allie, P., 1998, A Room of Your Own: What Would it Take to Help Remote Groups Work as Well as Collocated Groups? In CHI 98 018-23 APRIL 1998. ACM ISBN l-581 13-028-7

[21] Bekker, M., Olson, J. & Olson, G., 1995, Analysis of gestures in face-to-face design teams provides guidance for how to use groupware in design. In Proceedings of the Symposium on Designing Interactive Systems, DIS'95, 157- 166.

[22] Herbsleb, J. & Mockus, A., 2003, An Empirical Study of Speed and Communication in Globally Distributed Software Development. In IEEE Transactions on Software Engineering, Vol. 29, NO. 6, June 2003. pp. 481-494.

[23] Kobitzsch, W., Rombach, D., & Feldmann, R.L., 2001, Outsourcing in India (software development). Software, IEEE, Volume: 18, Issue: 2, March-April 2001. pp. 78-86.

[24] Collins, H., 1999, Regulating contracts. Oxford university press. ISBN: 0199258015

[25] Tiikkaja, M., 2002, Experience report: Case: COTS SW Component Acquisition and Management Process. Minttu project.

[26] Braun, P., 2003, Metamodel-based Integration of Tools. In Proceedings of TIS 2003 Workshop on Tool Integration in System Development. ESEC/FSE 2003 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering Helsinki, Finland September 1-5, 2003.

[27] Maznevski, M. & Chudoba, K., 2000, Bridging space over time: Global virtual team dynamics and effectiveness. Organization Science 11, 5 (May 2000), 473–492.

[28] Bjerknes, G. & Mathiassen, L., 2000, Improving Customer-Supplier Relation in IT Development. Proceedings of the 33rd Hawaii International Conference on System Sciences. 2000.

[29] Reifer, D., 2004, Seven Hot Outsourcing Practices. In IEEE SOFTWARE January/February 2004

[30] Coleman, D., 2000, Architecture for Planning Software Product Platforms. Tutorial presented at the First Software Product Line, Denver, Colo., Aug. 30–Sept. 1, 2000.

[31] Hansen, K., 2003, Activity-Centred Tool Integration Using Type-Based Publish/Subscribe for Peer-to-Peer Tool Integration. In Proceedings of TIS 2003 Workshop on Tool Integration in System Development. ESEC/FSE 2003 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering Helsinki, Finland September 1-5, 2003.

[32] Carmel, E. and Agarwal, R. 2001. Tactical Approaches for Alleviating Distance in Global Software Development. In IEEE Software March/April 2001. pp. 22-29.

PAPER III

Merlin collaboration handbook

The challenges and solutions in global collaborative product development

In: Proceedings of the Third International Conference on Software and Data Technologies. Porto, Portugal, 5–8 July, 2008. Special Session on Global Software Development: Challenges and Advances on ICSOFT 2008. Pp. 339–346.

Merlin collaboration handbook: Challenges and Solutions in Global Collaborative Product Development

Päivi Parviainen, Juho Eskeli, Tanja Kynkäänniemi, Maarit Tihinen

VTT Technical Research Centre of Finland,Kaitoväylä 1, Oulu, Finland
paivi.parviainen@vtt.fi, juho.eskeli@vtt.fi, tanja.kynkaanniemi@vtt.fi,
maarit.tihinen@vtt.fi,

Abstract: Global, collaborative and distributed development is increasingly common in software development. However, the traditional product and software development technologies do not support this way of working well, e.g., time and cultural differences cause new requirements for these technologies. In this paper, we introduce a public web-based handbook, collecting the challenges encountered in global collaborative development by the companies, and also a large number of solutions that help in tackling these challenges. The handbook was implemented using an ontology editor and generating HTML pages. In the final phase of the development the handbook was validated by several external testers, with main feedback being that the handbook was found useful, but more practical solutions would be welcome. Handbook was also updated based on the feedback.

1 Introduction

In the perspective of growing size and complexity of embedded systems, companies are not able to develop all the required functionality by themselves. As a result, suppliers specialize in specific functionality or specific skills which they can sell to others. This is clearly visible in the growing numbers of the outsourcing constructions in the past years. For example, a survey [1] found that 74% of the participated companies had more than one development location. 48% had four or more locations and 26 % had more than 20 locations. Furthermore, a major survey carried out by the Software & Information Industry Association (in January 2007) indicates that companies are increasing their global development efforts: 57% of the survey participants have significantly increased offshore work in the past 18 months and many plan to add still more in the next 18 months. Growth strategy was cited as an important or critical driver for 84% of respondents, while increasing speed to market and productivity were the next most important drivers. Collaborative engineering of embedded systems has become a fact of life, and currently there is no way back anymore; companies have already outsourced large parts of their developments to other companies, resulting in no longer having the related skills available in their own organisations. Instead, the companies need to manage a complex situation of many partners, sub-contractors, suppliers, software platforms and so on.

Practice has shown that the traditional single company development technologies do not support collaborative product development well. For example, another survey shows that 80% of companies are unsatisfied with their overall collaborative development efforts. Survey respondents expressed as main problems the poor foundation for collaboration and poor management of partner relationships. These problems are often caused by, e.g., time difference and geographical distribution that cause new requirements to the ways of working and tools. Also, understanding each other is not straightforward due to different backgrounds, e.g., different terminologies, cultures etc. but needs to be supported by technologies.

There are some experience reports about challenges companies have faced in collaboration, for example, Philips [2], Siemens [3], Motorola [4], Alcatel [5], and Lucent Technologies [6]. Also several books that are discussing the problematics in collaboration and solutions to address them have been published [7. 8. 9. 10 & 11]. There are also conferences and workshops such as ICGSE (International Conference on Global Software Engineering) dedicated to global software engineering. However, these sources cover only some viewpoints of collaborative software development and until now no comprehensive collection of challenges and solutions for product development in collaboration could be found. Still, all these sources have been used as input for the Merlin Collaboration Handbook.

In this paper collaboration means that two or more entities work together to create mutual value. These entities can be companies, departments or even teams and they can combine in any one of several different

business relationships and for very different periods of time. Importantly, the entities are physically in different locations, i.e., the development is distributed.

This paper is organized as follows: first, in chapter 2 we discuss the process of writing the handbook. Next we present the contents of the handbook in general level, including the structure and technical implementation of the handbook in chapter 3. In chapter 4 we discuss the validation of the handbook and end the paper with some conclusions and thoughts for further work.

2 Merlin Handbook Development

The purpose of the handbook - defined in co-operation with the Merlin project consortium - is to support operational collaborative development, i.e., help companies to take care of all critical aspects during various phases of the collaborative project. In practice this would be done by collecting, listing and structuring these critical aspects as well as ways to address them, called solutions into a handbook. Furthermore, in order to make the handbook usable, ways to access parts of the contents based on user's interests should be made easy.

The building of the handbook started by defining it's structure; an initial framework for the structure of the handbook was derived from literature. CMMI [12] was used as the basic structure due to its wide acceptance in software world and challenges reported by others grouped according to the CMMI structure. Based on the initial framework an industrial inventory was carried out, including interviewing the industrial partners of the Merlin project. These interviews lead to many refinements to the framework, especially in the details, although several of the challenges encountered by the interviewed companies were also mentioned in literature. We have discussed these differences in [13], main being that the approaches represented by literature, on one hand, and industrial practitioners, on the other, towards problems related to collaborative work are different. The industry emphasizes technical aspects and detailed problems concerning engineering practices, while the literature focuses on solutions for more general issues like communication and team building. We found plenty of solutions for management and support practices in the literature but only few solutions for engineering practices. Thus, in order to provide more content to the handbook, collection of best practices from Merlin industrial partners was also done via focused interviews on selected topics. Results of these focused interviews were then included as solutions and experiences related to them in the handbook. Finally, also the research and development work done during the Merlin project was added to the handbook as solutions.

In order to support usability, e.g., the different views, and due to very large amount of content, the handbook was not implemented as a physical book, but a structured documentation solution was used to support readability. Implementation is discussed in more detail in section 3.3.

In practice, the handbook was developed in bi-monthly workshops with the Merlin consortium to refine implementation and contents of the handbook based on prototypes. The workshops had also representatives with experience on such repositories and usability, for example. Also, in the last phase of the project, the Handbook was validated by 16 external testers (this is discussed more in section 4).

3 Handbook Contents

This section describes the contents of the handbook in general level. The complete handbook is available in the internet (http://www.merlinhandbook.org).

3.1 Structure

The handbook structure is presented in the following Figure 1.

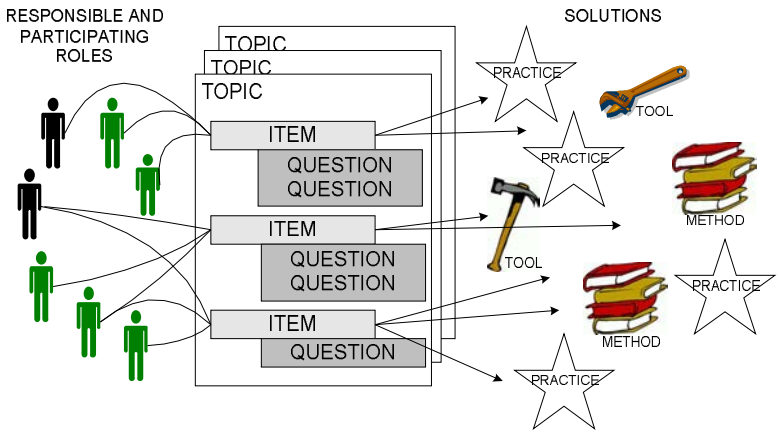


Fig. 1. Merlin collaboration handbook structure

The handbook structure is formed using topics; three main topics, namely management, engineering and support practices and 21 subtopics, such as collaboration management, requirements definition, testing, configuration management, and co-operative work. Each subtopic has number of important items, altogether about 80 of them, such as product roadmapping, conditions for collaboration, practices for resolution of conflicting requirements, sharing of test cases, unified CM practices, cultural differences, etc. Items are then still refined to questions, that further detail the item. For example, "Are supplier agreements and long-term framework agreements used as input for roadmapping?" is a question under "product roadmapping" item. The challenges faced by industry are especially visible in the items and questions of the handbook; they were first gathered in the industrial inventory and critically refined during the handbook development. The topics are mainly general, following CMMI structure to a large degree and the collaboration specific issues are visible in the item, question and solution level; we included only items, that are either specific for collaborative development or as in most cases are more difficult and complex to manage in collaborative situation.

In addition to topics, items and questions, also roles are defined. There are both responsible role and main participating roles defined for each item. The roles include all common product development roles, for example, senior manager, project manager, chief architect, etc. By roles, a checklist of questions the role is responsible for can be retrieved from the handbook. For example, Table 1 shows the checklist for the role Quality manager.

For each item also solutions are included in the handbook. Solutions are methods, techniques, tools, and practices that help in taking care of important items and they are classified according to their level of validation:

- Academic case, meaning that the solution is proposed in literature, often with academic case studies. These types of solutions were included also to provide ideas for items that were not so well covered with industrially proven solutions.
- Industrial case, meaning that the solution is proposed in literature or developed in Merlin with industrial case studies.
- Legislation or standards.

Each solution has also a standard description including ID, name, summary, description, evidence of suitability (level of validation), type of solution, collaboration dimensions, and references to further information.

3.2 Example of contents

An example of contents of the handbook is requirements engineering. In the handbook requirement engineering is divided to two topics, requirements development and requirements management. Requirements development has eight important items and requirement management has three important items defined in the handbook.

Table 1. Quality manager's checklist

A. Management practices	
Are relationships between common quality management process and partners own quality practices defined?	Responsible
Does the contracting process take into account all involved parties or stakeholders and do they have the required power of authority or signing?	Participates
Are suppliers or customers co-operation capability analysed beforehand?	Participates
Are the partners processes and quality management system of enough maturity?	Participates
B. Engineering practices	
Are the costs for non-quality taken into account while releasing the product. Have the costs for non-quality of the various suppliers been estimated?	Participates
Are performed tests and test results communication practices between partners defined and followed?	Participates
Are practices for incorporating feedback from customers to requirements development process defined and working?	Participates
Have the results of acceptance testing been taken adequately into account?	Participates
C. Support practices	
Is common process across sites or partners as thin as possible and forced as little as possible?	Responsible
Is the in-house review process defined and followed by each partner?	Responsible
Are best practices recoded and used between partners?	Responsible
Are common practices for quality assurance defined?	Responsible
Is shared process improvement work defined and agreed upon in long term relationships?	Responsible
Are common templates defined and used where applicable?	Participates
Is the effectiveness of collaboration evaluated for example as part of end-of-project evaluations?	Participates

An example of important item for requirements development is "Clear prioritization rules and practices / trade-off of the requirements". This item has five solutions in the handbook that help taking care of priorisation. These solutions are methods that base on giving values to different requirements (pairwise comparisons, e.g. AHP), negotiation approaches that base on the idea that the priority is determined by reaching agreement between the different stakeholders and dedicated requirements priorisation methods and techniques specifically supporting collaboration.

Another example of topic is collaboration management. For this topic nine important items have been defined, including for example, "establishing / evaluating conditions for collaboration" and "clear agreements with suppliers". The latter has four solutions, e.g., guidelines for acceptance criteria definition and creating a contract.

3.3 Technical implementation

Handbook data is stored and managed using Protégé. In general, Protégé is an open platform tool for ontology modelling and knowledge acquisition framework [14]. It offers a way to manage the cross-references in a mass of textual data in a RDF/OWL format [15]. There are predefined ontologies available on the web, which can be imported into Protégé and can then be used as a basis for other ontologies.

To develop the Handbook with Protégé, a data model of the Handbook was created. This was done by studying the structure of the Handbook (practices, topics, etc.), their relationships and the requirements for categorizing scopes. According to this study, the data model was designed. A decision was made to use the owl: Tool -ontology as a basis for the Handbook ontology. The Tool -ontology was chosen because it's content and link structure closely resembled that of the handbook. The Tool -ontology was then modified to reflect the Handbook data model. When the structure was finalised, the instances, that is, the content of the Handbook with their relationship information, was inserted into the ontology.

In order to be able to represent the Handbook data in web format, conversion from OWL/RDF format into something more suitable for our purposes was needed. This was done by using Protégé's possibility to extend

its functionality via plugins. A special purpose plugin was written which exports the OWL/RDF information into easily usable XML format. In the new format the data is structured as a tree, that is, on top there is a practice with its attributes, a practice has items with their own attributes and so on.

The next challenge was how to represent the information to users via web interface. Major requirement specified for the HTML Handbook was to have a possibility to scope the content according to user specifications in order to offer different views into the contents.

Scoped view into handbook was achieved by defining scoping parameters which can be used to bring out the different views. When entering the handbook, the user is first presented with a form in which he/she can tick suitable parameters to trigger the scoping process. Consequently, the same parameters are inserted in the Handbook ontology as attributes. These attributes make it possible to scope the content by using the XML data file generated by the Protégé converter and the Java servlets which ultimately render the selected content for the user. The following Figure 2 illustrates how handbook operates.

This approach makes the handbook content management simple; after the updated content has been defined in Protégé, it is only necessary to run the converter plugin and to deploy the new XML file to handbook web pages. This solution also guarantees that all the web pages have uniform layout.

From the usability point of view the scoping alone was not enough for navigating the handbook data. Therefore the handbook offers a search mechanism for its users by the means of Apache Lucene search-engine. Lucene offers many powerful features, most notable being the offline indexing support. Initially several other search engines were tested, but these were deemed too slow, mainly because they lacked the offline indexing feature and instead relied on to dynamical crawling. Because Lucene is available as open source it could be easily modifed to support the special features of handbook, namely the scoping.

The handbook was developed iteratively; bi-monthly workshops were arranged where the development versions of the handbook were presented to project members. Most of the feedback received from these sessions were improvements to the user interface (UI) and general usability, but also the scoping mechanism was defined in the workshops. These workshops were found to be especially important for the developers so they could see they were on the right track and could receive further guidance when needed. Also, they provided the project members continuous updates to current situation and opportunities to influence the Handbook.

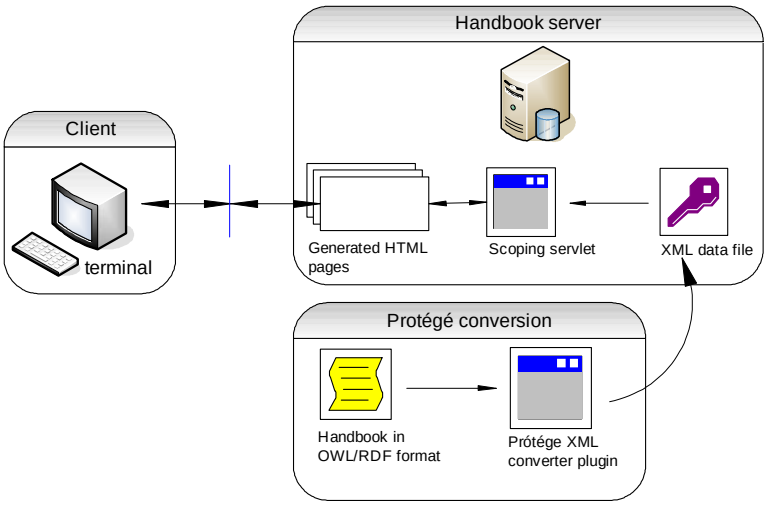


Fig. 2. Merlin Handbook data flow

4 Handbook validation

During the project, the handbook was validated in two different ways in addition to the workshops that gave continuous feedback for the development. Firstly, 16 external testers used the handbook and provided feedback and secondly the handbook was used in an industrial case to support improving subcontracting efficiency in a company. Both of these are discussed in this chapter.

4.1 External testers

In the final phase of the Merlin project, the handbook was tried out by external testers. These testers were found by asking the Merlin industrial partners to find persons in their organizations that have not been involved in the Merlin project but who are knowledgeable in the topic, i.e., collaborative product development. Another source for these external testers was the public seminars, where the handbook prototypes were presented and volunteers for testing asked for. The users were typically project managers that had experience in collaborative projects and altogether 16 external users were granted access to the handbook during development.

During the validation, the external users were asked to think of a typical problem they would have in collaborative product development and try to find answers and help from the handbook to address this topic. No further instructions on using the handbook were given at first. Also, the feedback was to be given in free format including the chosen problem and the findings and other comments.

Feedback from the external users related mainly to the usability or the content of the handbook. Based on the testers' experiences, the handbook looked nice and worked fine. Also, one of the testers noted that handbook had very clear page lay-out. However, some handbook mechanisms needed explanations or guidelines for use. For instance scope selection page was not self-explanatory (e.g., what was meant by item, type of source, agreement base, etc.). Also, bookmark mechanism was not explained, thus it was not clear where the bookmarks were accessible.

The comments to the content of the handbook, related to new solutions that should be added to handbook. As one of the testers reported, on many problem area's underlying documentation was not yet given. Thus, it was suggested that following information should be added to the handbook: reporting practices, multi-site peer reviews, selecting configuration management tools, interactive process model based on best practices, and data on measurements. Also, one of the testers reported that the handbook included too little information on measurements, metrics, reporting and follow-up in both management and project management. It was also requested that the handbook should answer to following questions: what are readiness assessments or checklists for collaboration, and why to work collaboratively?

Furthermore, according to one of the testers, the overall information in the handbook tended to be quite theoretical. Hence, for practitioners, practical or implementation examples were missing, meaning that findings remained abstract and theoretical. Some guidelines were provided in the handbook, but it could not be found how to do that in practice. It was also pointed out by another tester that the usage of proven practices was a good idea, since a larger number of examples would bring additional value to the handbook.

It was also pointed out by one of the tester's that references to Google and Scholar Google sites should be improved, e.g., to include the key words of the specific publication in the URL, so that the users wouldn't have to retype the words themselves. It was also noticed that common terminology for the handbook should be provided in order to avoid inconsistency of the terms used in the handbook.

Based on these comments by external testers, handbook usability and content has been improved. Now, the handbook is faster and easier to use. For instance, help texts and terminology have been added to the handbook in order to facilitate the use. Also, information is much easier to find, then before the users comments, since scoping and search operations have been added to the handbook. With the scoping operation, the content of the handbook can be shown based on user's needs and situations and with the search operation specific topics, items or solutions can be easily found from the handbook. Also, based on the external testers' comments and wishes, the content of the handbook has been improved, for instance, new solutions have been created. User experiences also affected to the content of the solutions, e.g., what attributes (i.e. geographical distance, cultural differences, time difference, etc.) are taken into account in the solutions. Also, solutions' references have been updated. Furthermore, at the end of Merlin project, all results from the project have been written in

solution format, so that the content of the handbook would be more extensive and to include more experience-based solutions than before, as the users requested.

4.2 Industrial case

The Merlin Handbook was used to first analyse and then to improve the subcontracting practices of a company participating in the Merlin project. The aim of the case was specifically to improve controllability and efficiency of the subcontracting. The Handbook structure of items and questions was used as interview framework to find out the strengths and weaknesses of the current practices. This resulted in several improvement areas for the current practices but also to some additions to the Handbook items, as some challenges identified were not yet included in the Handbook. Then the handbook was used to find solutions to the improvement areas. Several solutions were found and were then applied to the company's needs.

The Handbook helped especially in gaining confidence to change as the solutions and experiences presented in the handbook supported the company's own ideas. Use of handbook helped also in minimizing risks, as the handbook could help in providing proven guidelines that can then be applied to own situation. Also, instead of having to reinvent the wheel the knowledge gained by larger network of people could be utilized. That saved time, due to avoiding using effort on basic issues and being able to focus on adapting proven solutions to own needs.

As a result, due to the improved, more effective practices, the number of the subcontracted personnel could be significantly increased, meaning that more work can now be subcontracted, freeing the company's own personnel to other tasks.

5 Conclusions and further work

In this paper we have introduced a public web-based handbook, collecting the challenges encountered in global collaborative development by the companies, and also a large number of solutions that help in tackling these challenges. The handbook was implemented using an ontology editor and generating HTML pages. In the final phase of the development the handbook was validated by several external testers, with main feedback being that the handbook was found useful, but more practical solutions would be welcome. Handbook was also updated based on the feedback. The Handbook was also found useful in improving a company's subcontracting practices, especially as it provided confidence to change.

The handbook is a collection of both literature and industrial experience. Especially the structure of the handbook, the topics, items and questions are based on challenges faced by industry. We have then made a collection of available solutions to these challenges and provided sources for further information. Many of the solutions are based on industrial experience, however, this is always situation dependent and it is up to the user to consider the usefulness of the solution to his/her situation. Also, the topic and different usage situations to be covered by the handbook is very large. Thus, we realize we could not cover everything during this three year project. However, based on the feedback from the users of the handbook so far, we can say that the handbook is helpful for a company working in collaboration with others; it can at least give ideas and triggers to consider while doing the work, and even provide complete, validated solutions to tackle the faced challenges.

In order to further advance the handbook, we have included an opportunity in the handbook to add new solutions by the users. However, these new solutions will first be reviewed by the Merlin project partners before they are accepted to the handbook. We also welcome other feedback. The handbook is publicly available from http://www.merlinhandbook.org.

Acknowledgements

The work described in this paper has been carried out in the Merlin[1] ITEA[2] project in co-operation with the whole consortium. The authors would like to thank the Merlin project members for the active participation in the handbook development.

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

References

1. VA Software, 2005, Application Development and Open Source Process Trends: Survey Analysis and Findings, white paper, January 2005, available from: http://www.vasoftware.com/gateway/pollresults.php
2. Kommeren, R., Parviainen, P.: Philips experiences in global distributed software development. Empirical Software Engineering Journal. Vol. 12, No. 6 -- 647-660 (2007)
3. Bass, M., Paulish, D.: Global software development process research at Siemens. In: The 3rd international workshop on global software development, May 24, 2004, In proceedings of ICSE 2004, International Conference on Software engineering, Edinburgh, Scotland, May (2004)
4. Battin, R.D., Crocker, R., Kreidler, J., Subramanian, K.: Leveraging Resources in Global Software Development. IEEE Software, March/April 2001, pp 70–77 (2001)
5. Ebert, C,, De Neve, P. : Surviving global software development, IEEE Software, March/April 2001, pp 62–69 (2001)
6. Herbsleb, J.D., Mockus, A., Finholt, T.A., Grinter, R.E. An empirical study of global software development: distance and speed. In the Proceedings of 23rd International Conference on Software Engineering, IEEE, Toronto, 2001.Also in. IEEE Trans Softw Eng 29(6):481–494, June 2003 (2001, 2003)
7. Karolak, D.W.: Global Software Development: Managing Virtual Teams and Environments, Wiley-IEEE Computer Society Pr; 1st edition (December 27, 1998)
8. Carmel, E.: Global Software Teams: Collaborating Across Borders and Time Zones, Prentice Hall (December 1998)
9. Sahay, S., Nicholson, B., Krishna, S.: Global IT Outsourcing: Software Development across Borders, Cambridge University Press (January 12, 2004)
10. Carmel, E., Tjia, P.: Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce, Cambridge University Press (June 13, 2005)
11. D. Damian. Stakeholders in Global Requirements Engineering: Lessons learned from practice. IEEE Software, Mar/Apr 2007.
12. CMMI for development, version 1.2., Technical Report CMU/SEI-2006-TR-008. (2006)
13. Hyysalo, J., Parviainen, P. and Tihinen, M.: Collaborative Embedded Systems Development: Survey of State of the Practice. Proceedings of 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006). pp. 130–138. (2006)
14. Protégé web pages: http://protege.stanford.edu/
15. RDF pages: RDF/XML Syntax Specification http://www.w3.org/TR/rdf-syntax-grammar/.

[2] http://www.itea-office.org

PAPER IV

Knowledge related challenges and solutions in GSD

In: Expert Systems. The Journal of Knowledge Engineering (2011).

DOI: 10.1111/j.1468-0394.2011.00608.x

Article

DOI: 10.1111/j.1468-0394.2011.00608.x

Knowledge-related challenges and solutions in GSD

Päivi Parviainen and Maarit Tihinen

VTT Technical Research Centre of Finland, Finland
Email: paivi.parviainen@vtt.fi

Abstract: *A number knowledge-related challenges may complicate the work in global software development (GSD) projects. In practice, even a small amount of missing knowledge may cause an activity to fail to create and transfer information which is critical to later functions, causing these later functions to fail. Thus, knowledge engineering holds a central role in order to succeed with globally distributed product development. Furthermore, examining the challenges faced in GSD from a cognitive perspective will help to find solutions that take into account the knowledge needs of different stakeholders in GSD and thus help to establish conditions for successful GSD projects. In this paper, we will discuss these challenges and solutions based on an extensive literature study and practical experience gained in several international projects over the last decade. Altogether, over 50 case studies were analysed. We analysed the challenges identified in the cases from a cognitive perspective for bridging and avoiding the knowledge gaps and, based on this analysis, we will present example solutions to address the challenges during the GSD projects. We will conclude that through understanding both the nature of GSD and the KE challenges in depth, it will be possible for organizations to make their distributed operations successful.*

Keywords: knowledge engineering, global software development, industrial challenges

1. Introduction

Trends in global product developments show that the size and complexity of software intensive systems will continue to grow, making it difficult for companies to develop all of the required functionality alone (van Solingen *et al*., 2008). Thus, the products are being increasingly developed in a globally distributed fashion (Carmel & Agarwal, 2001; Hyysalo *et al*., 2006; Noll *et al*., 2010). At the same time, several papers have reported that software projects miss their schedules, exceed their budgets, and deliver software products of poor quality or in the worst cases, even with the wrong functionality (The CHAOS Reports, 1996, 1998, 2000, 2002, 2004 and 2006; Olson & Olson, 2000; Herbsleb *et al*., 2001; Damian & Zowghi, 2002; Bhat *et al*., 2006; Jimènez *et al*., 2009). Furthermore, the traditional product and software development technologies (practices, processes, tools) do not support globally distributed product developments well. For example, the development teams are usually dispersed geographically and this alone causes new requirements for used technologies; such as higher demands on communication and teamwork methods (Aranda *et al*., 2006; Layman *et al*., 2006). Furthermore, in global software developments (GSDs), different time zones and distances make communication more difficult than in a local (single-site) development. The physical distance between the development sites alone, causes problems in task coordination, project

management and communication tasks (Olson & Olson, 2000; Carmel & Agarwal, 2001; Herbsleb & Moitra, 2001; Damian & Zowghi, 2002; De Souza *et al*., 2002; Herbsleb, 2007; Jimènez *et al*., 2009). Moreover, the understanding of each other is not straightforward, due to different backgrounds in the terms of terminologies and cultures (Komi-Sirviö & Tihinen, 2005; Noll *et al*., 2010). Thus, the whole product development process differs significantly from the local development process and everything needs to be supported by technologies.

The role of knowledge and knowledge engineering (KE) is crucial in product developments, but it is even more important in global product developments due to distance and cultural aspects, for example. Davenport and Prusak (1998) describe knowledge as a dynamic blend of experience, values, contextual information and expert insight. This kind of knowledge provides a framework for evaluating and incorporating new experiences and information. Furthermore, Noble (2004) points out that a cognitive perspective is a fundamental factor of success for teams in collaboration. He describes both, the kind of knowledge which is important to team effectiveness, and how teams employ this knowledge to coordinate, make decisions, and achieve consensus. Thus, KE and knowledge-related challenges hold a central role, while developing solutions for supporting globally distributed product developments.

This paper introduces knowledge-related challenges in GSDs that need to be understood and addressed in order to enable the success of the GSD projects. Some practical solutions (methods, practices, tools) that have been developed to overcome these problems are also described in the paper. The discussed challenges and solutions have been gathered during research performed in several international projects at VTT Technical Research Centre of Finland (VTT, 2011) over the last decade. We argue that the examination of challenges from a cognitive perspective will help to establish solutions that take into account the knowledge needs from the viewpoint of different stakeholders in GSD.

2. Background and related work

The terms, data, information and knowledge can be understood to be overlapping concepts. When considering their levels of abstraction, data is the lowest level, information is the next level, and knowledge is at the highest level of the three. Data are defined to be facts about events, without any interpretation about the event. Information can be described as a message from a sender to a receiver. The main purpose of information is to have an impact on the receiver's judgement and behaviour. Davenport and Prusak (1998) pointed out that knowledge is more; it is a dynamic blend of experience, values, contextual information and expert insights. Nonaka (1994) distinguished these two dimensions of knowledge: explicit and tacit knowledge. Explicit knowledge is understood to be knowledge that can be articulated, codified or stored in a certain media. It can also be transmitted to others. To a large degree, tacit knowledge is knowledge that cannot be articulated. In the knowledge management (KM) domain, the conversion of tacit knowledge to explicit knowledge is seen as a highly critical process, since tacit knowledge consists of such habits and culture that we do not possess ourselves. This means that an effective and successful transfer of tacit knowledge requires extensive personal contacts and trust.

Davenport and Prusak (1998) define knowledge as follows: 'Knowledge is a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of knowers. In organizations, it often becomes embedded, not only in documents or repositories but also in organizational routines, processes, practices, and norms.' Thus, knowledge can and should be evaluated by the decisions or actions to which it leads. For example, improved knowledge increases the effectiveness of product developments and production. Knowledge can be used for making wiser decisions about strategies, potential customers, main competitors, distribution channels,

products and services (Davenport & Prusak, 1998).

2.1. KE for bridging knowledge gaps in GSD

KE is a relatively new branch of software engineering. KE is an evolutionary process of engineering artefacts and using them to gain new understandings, and these new understandings are then used to further engineer or modify artefacts, whereupon the process continues. Over recent years, common awareness has been created about that, a strong interplay exists between software engineering andKE, and studies have been directed as to how KE methods can be applied to software engineering, and vice versa. Noble (2004) illustrated the relationship between knowledge ('Individual and Shared Understandings') and some key team activities, as shown in Figure 1.

All teams perform all of the team activities described above, generally moving from left to right, but also switching back and forth among the activities, depending on their immediate needs. According to Noble (2002), the team leader analyses the mission and determines the required members and resources, and then recruits the team, and assigns tasks and resources during a 'set up and adjustment' process. The team revises its set up whenever members decide to change their team organization, tasks, or infrastructure. Some of this knowledge can be written down, but a large amount will remain as tacit knowledge in the minds of the team members. They will need this knowledge while carrying out their 'group problem solving' process, while team members may brainstorm, critique and enrich, evaluate and prioritize, discover differences, negotiate, reach a consensus, identify solutions, and make decisions. In 'synchronize and act', they draw on their knowledge to coordinate and help each other. This coordination enables the team as a whole to realize the benefits of teamwork. These include the enabling of task continuity over time and space through coordinated handoffs, increasing physical impacts through the massing of effects, improved efficiency by team members laying

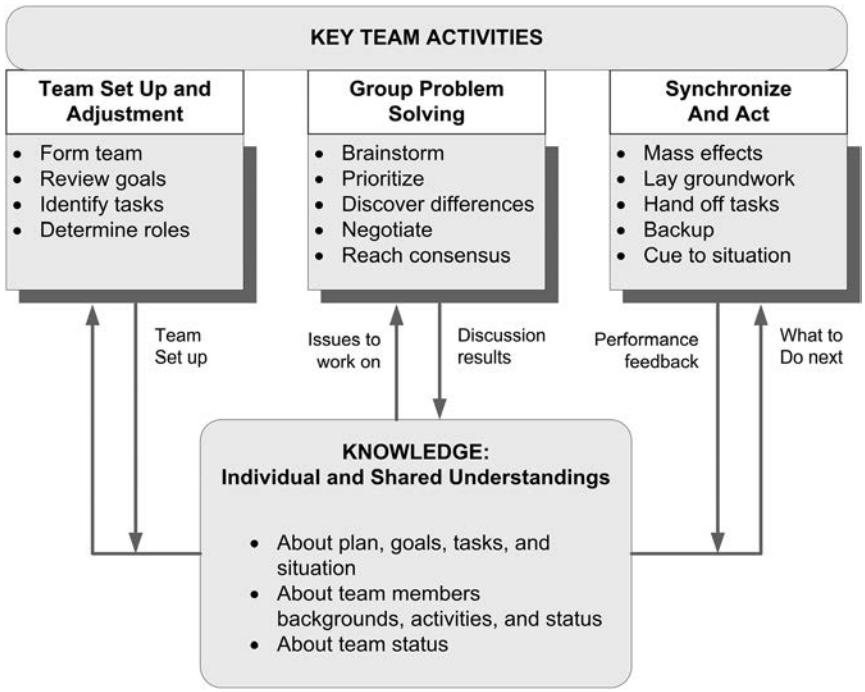


Figure 1: *The relationship between Knowledge and Team Activities (Noble, 2004).*

the groundwork for each other, and an increased reliability by team members backing each other up. Figure 1 shows that any knowledge gap within the team can grow into larger problems and may lead to the poor sharing of information or lack of knowledge of what to do. Software development is a very knowledge-intensive field of engineering, as in each development phase, efficient knowledge creation, knowledge transfer, knowledge storing and/or knowledge sharing activities are vital. Thus, all of the problems faced and the challenges in GSD should be analysed from a cognitive perspective for bridging and avoiding the knowledge gaps. This analysis will also enable the identification of the best available solutions to the challenges during the work.

2.2. *The role of knowledge in GSD*

Several articles have been published where KM based challenges have been discussed in more detail. For example, Rus and Lindvall (2002) provided an overview of over 40 submitted papers presenting the Software Engineering applications of KM. Furthermore, Rus and Lindvall (2002) and Desouza *et al*. (2006) introduced a large number of knowledge needs and challenges during the software development. They also present how these activities should be systematically approached in the context of distributed software development, via a proposal that an organization must construct a concerted global KM strategy. Rus and Lindvall (2002) discuss the importance of individuals having access to the correct information and knowledge when they need to complete a task or make a decision. Knowledge must be managed in all the stages of software development: from the encapsulation of requirements, to the creation and testing of a program, to the software's installation and maintenance and even extending to the improvement of organizational software development processes and practices. These tasks are more complicated in a distributed development than that which is local. Also, Damian and Moitra (2006) point out several improvement areas in GSD that are KE related, including KM strategies, distributed software development, requirements engineering, distributed requirements, and managing offshore collaborations. However, there are only a small number of papers where knowledge-related challenges in GSD have been discussed. Furthermore, most of these papers focus on building a KM system or a strategy for an organization, or on the other hand, the introduction of experiences gathered from a tool-based solution of sharing information, experiences or documents, inside and over the projects. The importance of socio-technical or cognitive aspects of the challenges identified in GSD were only discussed in a few articles (Aranda *et al*., 2006; Noll *et al*., 2010). In this article, we will focus on introducing the challenges faced in the industry during a global product development, how these challenges are knowledge related, and what kind of solutions are available to solve these challenges. We will use the model of Noble (2004) for identifying and analysing the knowledge needs of distributed teams and stakeholders. This approach increases the visibility of knowledge based requirements and challenges, thus making it possible to take them into account while carrying out improvement actions, and utilizing general KM solutions more extensively to solve GSD challenges.

2.3. *Research design*

Our research results, presented in this paper, are based on the following main sources:

- A survey (Komi-Sirviö & Tihinen, 2005) showed that the challenges of distributed software developments must be recognized when the objective is to minimize the chance of development failures and maximize the possibilities for success.
- The MERLIN (2004–2007) ITEA project, where a more detailed study about the problems and solutions for collaborative SW development was carried out. During the project, several industrial cases were also performed, aimed at improving GSD in the participating companies.

- The PRISMA (2009–2011) ITEA2 project, where an update of both the state of the art and the state of the practice was made and further industrial case studies are carried out.

First, we used a questionnaire to survey the most problematic areas and knowledge based challenges in distributed software development (Komi-Sirviö & Tihinen, 2005). The semi-structured questionnaire was posted to 44 organizations in Finland and it was also e-mailed to over 200 organizations around the world (the questionnaire was also accessible via the Internet). The total number of responses was 31, representing 21 different organizations. This survey established a base for further research investments.

During the MERLIN project, we examined the most critical issues related to collaboration work and identified the most important areas for future research activities. The results of the study were published in Hyysalo *et al*. (2006). The study was carried out by performing interviews and reviewing existing material, including the process descriptions, templates, and guidelines, of the companies participating in the MERLIN project. The interviews were carried out using a specific framework. A total of 12 interviews of senior managers, project managers, software developers and testers from six different companies were carried out. The industrial partners represent several divergent embedded SW business areas: mobile and wireless systems, data management solutions, telecommunications, IT services, and consumer electronics.

A case study research method was used for the creation and trialling of new practices or other kinds of solutions against identified challenges and problems in collaboration. According to Yin (2003), a case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and the content are not clearly evident. During the MERLIN and PRISMA projects, from 2004 to 2010, a total of 54 industrial case studies were carried out. In our context, an industrial case means a trial of a new or enhanced practice, method, technique or tool(s), carried out in industrial settings, i.e., in product development projects. Each case study has been documented in a structured way as an experience report. In addition, a literature search was performed to find experiences and solutions published by others.

Thereafter, we have studied and analysed all 54 cases with respect to the knowledge intensiveness of the addressed challenges and tried solutions. Although all of the cases were somewhat knowledge related – as all activities in product developments are – we identified 40 cases from 12 different companies that were intensively knowledge related. In this paper, we have grouped these 40 cases into the following classes: (1) requirements engineering, (2) architecture and design, (3) integration and testing, (4) management and (5) support practices. The classification was made to assist the facilitation and clarification of the presentation of the results. After that, we summarized and identified challenges in GSD, according to the key team activities introduced by Noble (2004).

Finally, we have collected solutions to address the challenges identified in the cases. These solutions have been tried out in the industrial cases, and have often been presented in literature by others. In this paper, the solutions are presented using the Noble's model for emphasizing the knowledge needs of each perspective. More solutions (including these and more details for them) are described in the MERLIN Collaboration Handbook (2007) – a collection of the best practices that support collaborative software developments (Parviainen *et al*., 2008). The background for the handbook was literature, and the surveys and industrial cases carried out during the MERLIN project. For example, Philips' experiences and lessons learned over 10 years of global distributed development at Philips, derived from about 200 projects (Kommeren & Parviainen, 2007), were included in the collaboration handbook. During the PRISMA project, the collaboration handbook has been further developed and a new wiki-based implementation is being developed. In total, in the current version of the wiki, the solutions are

based on more than 130 published scientific articles. The solution descriptions are based purely on the industrial partners' experience (30%), purely on literature (50%) and a combination of both experience and literature (30%). However, the solutions are partly overlapping, i.e., separate solutions can have similar topics, and thus, more than the 30% of the solutions are addressed both in literature as well as in the industrial cases.

3. Knowledge-related challenges in GSD

In this section, knowledge-related challenges in GSD are introduced. First, the challenges that came up based on the surveys carried out in the projects mentioned earlier are presented. Then the challenges from the industrial cases are discussed according to the product development activity that they are part of. Finally, the identified challenges are discussed via their knowledge based perspective based on the Noble's key team activities.

3.1. Challenges based on surveys

The survey (Komi-Sirviö & Tihinen, 2005) was conducted, relating to knowledge based challenges in distributed software development. The survey results showed (Figure 2) that the most problematic area was tools and the environment, and more specifically, the incompatibility of the tools and versions used by the different development sites. This problem was emphasized most by large organizations employing more than 500 persons.

Problems relating to communication and contacts appeared to be very common within all of the organizations; this problem area was ranked as the second toughest. A closer analysis of the responses showed that the role played by communication was even greater than it appeared at first: the lack or poor quality of communication was often mentioned as a root cause behind other problems. One respondent described the problem: '*Sometimes, the level of English does not even allow for phone-conferences*'. In addition, requirements engineering (RE) appeared highly problematic for distributed development projects, causing a large number of errors (Komi-Sirviö & Tihinen, 2005).

Another study about the problems and their solutions in collaborative SW development was carried out during the MERLIN project. The main challenges in the collaboration, as seen by the partners, varied including:

- The openness of communication between partners, e.g., problem hiding may be an issue.
- Unclear assignments/specifications of the work in contracts and establishing good understanding between the partners concerning each others work: When all of the collaboration partners have the same view/a shared understanding of what is to be done

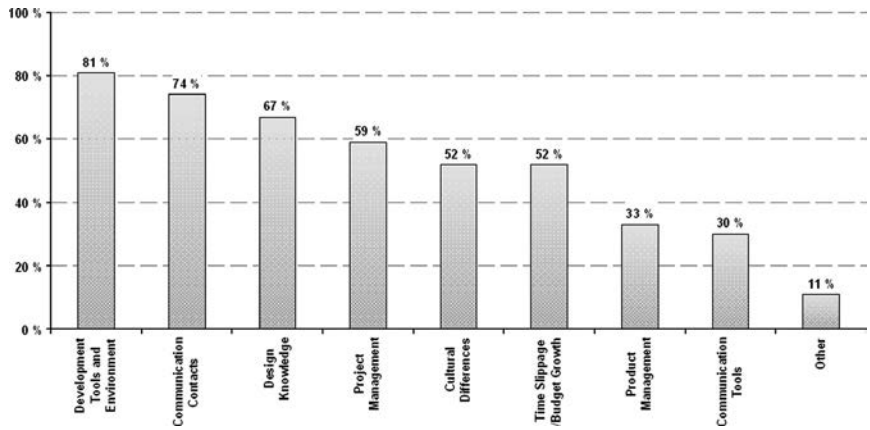


Figure 2: *Problem areas in distributed SW development (Komi-Sirviö & Tihinen, 2005).*

and if that's written down well, fewer conflicts will occur.
- Trust between the partners. If trust is not there, more formal practices for a follow-up need to be applied, resulting in more work.
- The reliability of the partners' development schedule, especially when there are dependencies in the partner's work.
- A real need for co-operation, that is, a mutual benefit from the collaboration. Partners who complement each others expertise makes, e.g., agreements concerning the sharing of work and decision authorities easier.
- Becoming too dependent on one partner, e.g., when a partner has strong competence in something you do not have in-house, it is essential to accurately prioritize that partner's work, for example, in order to get the required features in the partner's future releases as there may not be any other way to get those features into the product.

3.2. *Challenges based on industrial cases*

In this subsection, the challenges based on industrial cases are introduced according to their main activity area in collaborative SW development. The identified challenges are discussed in detail from KE viewpoint.

3.2.1. Requirements engineering Requirements engineering contains a set of activities for discovering, analysing, documenting, validating and maintaining a set of requirements for a system (Sommerville & Sawyer, 1997). The analysis is based on 16 cases in 11 different companies as well as several workshops arranged in the PRISMA project.

Requirements gathering and prioritization: Several challenges have been identified relating to requirements gathering from various stakeholders, high level analysis and the prioritization of requirements by product management, and transferring the requirements to R&D. For example, improving the understanding of the customer needs, and the requirement acquisition and recording methods to improve the quality of recorded information have been addressed. These were seen as important topics in order to enable detecting when insufficient knowledge of the application and needs could result in wrong decisions and design errors. These challenges are very knowledge intensive, for example, knowledge is needed of the relevant stakeholders and their importance, so that the loudest do not automatically obtain the highest priority. As a company stated: 'There often seems to be more ideas and possibilities for a new product than what is feasible. Among other things, there are numerous internal stakeholders who view the market from different perspectives, customers have their own priorities and competition always needs to be regarded.' Also, communicating priorities to other sites is important, so that the work is performed based on correct priorities. Furthermore, describing requirements so that they are understood similarly by all stakeholders – with different backgrounds and tacit knowledge – is important, but challenging.

Requirements traceability: Requirement traceability means identifying requirements and then following their lifecycle, both forwards and backwards (Gotel & Finkelstein, 1994). Distributed development brings additional challenges to creating and maintaining the traceability, as it may need to be performed over company borders and to various tools. Traceability is important for providing information to change management, for example, for analysing the impact of a change proposal, as it is easier to define which modules and tests are affected when a change is accepted. Traceability has been addressed in the cases from the viewpoint of establishing and automating requirements traceability. In order to manage the traceability, one requires knowledge about how things are related to each other. This requires knowledge about the product structure and the development process artefacts, for example. A good management of traceability is important, so that the work is performed based on correct information, when the background understanding of the people involved is not necessarily the same.

Requirements communication/transfer/flowdown: Requirements communication, transfer and flowdown mean describing the requirements

so that they are understandable for others, transferring them to other partners, and flowing them down to subsystems. A common challenge that has been addressed in the cases has been to improve requirements documentation practices. A company expressed: 'We have trouble with requirements being interpreted, when the definition process is distributed.' Good requirement descriptions are very important in GSD, as they are important means of sharing information, e.g., the work performed by different sites/partners is often based on the requirement documents. People from different cultures and backgrounds do not necessarily understand the things the same way, so it is essential that requirement descriptions are unambiguous, consistent and clear. For example, in a company, a challenge was stated as follows 'Our subcontractor does not always ask for clarifications of unclear requirements, but instead, they have invented their own solutions that have subsequently not fitted with the rest of the product.' On the other hand, the time and resources available for the requirements definition are limited, so a challenge is to know the correct level of requirement descriptions. Also, ensuring the common understanding is challenging, as partners may be unwilling to communicate the unclear issues, or they are not aware of the different interpretations of the requirements until late in the project. As a company stated: 'It is challenging to validate each stakeholder's interpretation of requirements before the implementation takes place. Validation is typically performed with the delivery of a prototype or early build; this may result in wasted time and effort.' This is all very knowledge intensive; it requires proper knowledge creation and specifically a correct transfer of knowledge in the distributed development situation.

Several cases addressed challenges relating to non-functional requirements, often referred to as the qualities, for example, usability, maintainability and performance. Few cases have addressed the challenge of what methods and tools can be used to handle non-functional requirements in a multi-partner embedded software project. Non-functional requirements are vulnerable to different kinds of interpretations, which are more likely in GSD, due to different backgrounds of people. Thus, describing non-functional requirements well is even more important in GSD, so that they will be taken into account in everyone's work.

3.2.2. Architecture and design In this section, we will discuss challenges based on five cases in five different companies. A common challenge has been to establish a good architecture for a product or product-line. The design of good architecture is a very knowledge intensive activity. As one of the purposes of architecture description is to facilitate communication, similar challenges apply as with requirements. Establishing a common understanding over sites/partners is challenging due to different backgrounds, for example. Architecture should also be designed so that it supports the division of work to the various partners, which requires knowledge of the partners' capabilities and product requirements. The working culture may cause architectural differences in collaboration as the architectural views for problem solving can differ quite a great deal, for example, due to different foci (e.g., efficiency vs. implementation).

A related challenge has been to define a 'good enough' level of design in order to result in a reasonable level of documentation, while still providing the necessary information to all stakeholders, i.e., to detect a too little design or 'analysis paralysis'. In order to define what is necessary information in the design documents, knowledge about the stakeholders and their work is required. This is especially important in GSD, as the role of documentation is more important in knowledge sharing than in a single site. Furthermore, it is more complicated to define the relevant information for the stakeholders that are not so well known to you, and that can have different backgrounds and tacit knowledge. Some cases have also focused on validating the current architecture to improve the product architecture itself from defined viewpoints, e.g., the adaptability of the architecture: 'How software systems can adapt

to multiple platforms at an architecture level and how adaptability mechanisms can be added to architectures of software systems developed in collaborative work?' Good architecture and communication about the architecture are important, so that the parts which are made in different sites can be integrated together well, and so that there is no duplicate work or areas which are not covered.

3.2.3. Integration and testing Four cases from five companies (one shared case between companies) have addressed integration and testing. Several cases addressed integration issues, such as when the integration of the software takes place at different locations, it often finally lead to a non-buildable product, or as stated by a company: 'We have problems at integration, when remote programmers throw their build code ''over the wall'' to a build manager who must resolve conflicts'. Also, the required expertise and its availability during integration have been addressed in the cases. From a KE viewpoint, it is important to make sure that the integrator has enough competence, when product parts are made in remote sites and the integrator does not have continuous visibility with the work. Also, the developers should know that their work is only done when the product is integrated.

Some cases have addressed challenges related to that testing in a collaborative embedded software development is perceived to be inefficient, taking a too high portion of the total development effort. These cases have focused on developing common test practices (including test sets, and tool environments) usable for distributed projects. For example, extra effort can be caused by, e.g., repeating the defects due to the non-transparent and different view of the status of the software, due to working with tools which are not probably connected. Another example is related to sharing information: 'Tests done and their results are not known by the component provider's customers that run their own regression tests with their test data. I.e., the customers have limited knowledge of the tests already run and the impact of the changes made vs. previous versions, thus resulting in overlapping tests and duplicate work.' From a KE viewpoint, sharing information about the test plans, test environment, and the tests that have been carried out between partners is important to avoid duplicate work. Defining effective testing for a distributed project requires knowledge of the product, work distribution and schedules, test methods etc. and the discipline and tools to share the information between the partners.

3.2.4. Management The discussion in this section is based on 12 cases from 10 companies. In a distributed development, significantly more effort is required for up-front planning and follow-up activities in order to be able to manage a project successfully. The manager in a GSD project has to have a large amount of abilities and knowledge in addition to technical competence, such as cultural knowledge and communication skills and particularly good project management capabilities. As a company stated: 'In GSD, a project manager may be far away from the development groups, which creates a visibility problem, and makes it easier to hide problems.' In other words, distribution makes the project progress more difficult to estimate and control due to the decreased visibility.

Identification of the dependencies between partners – e.g., the interdependencies of the subsystem deliveries – and taking them into account in project schedules was seen as a critical issue. The dependencies should also be made explicit, by defining the responsibilities for the delivery (who, what, when, to whom), the authority to accept, as well as the acceptance procedure. The status of the dependencies should then be checked pro-actively.

Communication and information sharing: The challenge of sharing information and knowledge about the ongoing projects and other related information in GSD was also addressed in four industrial cases. Communication with the peers located on different sites was mentioned as a specific challenge. Additionally, the diminished contact with a dedicated project owner meant that the project team did not possess sufficient vision or one-on-one guidance to make important

design choices during the development. There were also knowledge based goals in the analysed industrial trials, such as improving the collaborative skills in projects development, and identifying problems which occurred during the case project related to the supporting tools, process and communication.

Resource Management: From a KE viewpoint, resource management is specifically challenging in a distributed development, for example, knowing what expertise is available in different sites over time requires specific attention (e.g., being aware of changes in project schedules and resource loads, when some expert can suddenly be available for other projects etc.). In practice, project managers prefer to use known resources – people they know to be good or experts in the topic, instead of finding out the available resources from other sites. This may result in the sub-optimal use of resources and expertise in projects.

Measurements and analysis: In GSD, it is important to get real-time and accurate information on projects while the work is performed in different sites or even by different companies. Two cases (from two companies) addressed measurements and analysis challenges in a collaboration situation. In both cases, KE was recognized to be in a vital role: in the analysis and interpretation of the measurements, knowledge sharing and lessons learned have to be taken into consideration, in order to make correct conclusions from the data.

Subcontract management: Subcontracting is a very typical activity in GSD and many of the challenges which are described in previous sections are also relevant in subcontracting. Five industrial cases from two companies have been carried out, where subcontracting practices were the targets of improvement actions. Generally, the cases focused on strengthening the companies' subcontracting practices or finding new practices for carrying out subcontracting, in order to improve the subcontracted outputs as well as the controllability and efficiency of the subcontracting. As a company stated: 'We have noticed that the subcontracting R&D projects in the HW SW development area is a challenging issue. SW is an abstract thing which can be difficult to specify comprehensively. The synchronization of simultaneous HW and SW projects, so that they are ready to be integrated on time, is quite demanding. The different backgrounds of project members and the distance of locations and the time difference can be significant. These things cause extra challenges in projects where the technical content itself is demanding. Therefore, it is not surprising that misunderstandings, delays, conflicts etc. can happen in collaboration projects.'

KE and management were considered to be highly significant aspects. Firstly, knowledge holds a major role when selecting a subcontractor: a company can complement its own knowledge via subcontracting or a company can decide that some knowledge will be outsourced. Second, subcontracting management is very knowledge intensive: differences in skills and knowledge between the partners need to be managed, and real-time and exact information sharing has to be ensured. In practice, the effort required by the subcontracting party to manage the subcontractor has often been underestimated: 'We had underestimated the time and effort that would have been needed from our own people to monitor and guide the subcontractor. As that time was not allocated, the subcontracted work was not as we had hoped.' Proper subcontracting management is exclusively possible if KE aspects such as knowledge gathering, transferring and sharing have been addressed.

3.2.5. Support practices Support practices mean all those activities that occur as ongoing or cross-section practices during a project's lifecycle. Several cases were somehow related to the support practices in collaboration. In three cases from three companies, KE aspects were in a major role. One case focused on ensuring effective configuration management in a situation where the work was distributed based on development phases and the project involved people of various backgrounds. There were, for example: 'Agreeing about the configuration management tool was difficult, as there were sites with different backgrounds and preferences. It is clear that selecting a configuration management tool and practices causes a great deal of sentimental arguing, some people like a certain tool and others some other and there is often no

real factual reasoning.' Another case was focused on the defect management process: 'Reporting about the defects found during product development has been troublesome, as there are no general guidelines or common tools for doing that. Communication is performed via email and by the phone between the resellers, integrator and subcontractor.' In the third case, intellectual property right (IPR) management was addressed, especially from the communication and agreement of IPR in the GSD viewpoint. All of these cases are knowledge intensive, as they involve sharing information that can be interpreted differently due to the different backgrounds of involved people. Thus, the practices related to these topics should be defined utilizing KE principles.

3.3. The summary of challenges

In this section, we will summarize the identified knowledge-related industrial challenges by presenting the knowledge needs from the viewpoint of the key team activities illustrated by Noble (2004). Noble's model was used since it was the best model that we found concerning knowledge intensive software production from a cognitive perspective, whereas, the models which are presented in literature usually focus on the KM and strategy viewpoint. Noble's model illustrates activities for effectively identifying knowledge needs and sharing knowledge during the collaboration in practice. This way, the cognitive perspectives of the challenges can be better perceived, enabling the identification of solutions to the challenges.

3.3.1. The challenges for 'Team Set Up and Adjustment' activities 'The Team Set Up and Adjustment' covers activities such as team forming, goals reviewing, tasks identifying and roles determining that have to be continuously updated to reflect the new knowledge which has been gathered and shared during GSD. In the following table (Table 1), the identified knowledge intensive challenges will be presented, along with examples from cases.

The factors behind the challenges were identified as described in Figure 3. Factors are the causes of the challenges and can be addressed with practices that take them into account. The relation of the factors to the activities defined by Noble is also shown in the figure, so that the KE solutions can be identified to address these factors and thus the challenges.

In GSD, the partners' tacit knowledge, relating to the different backgrounds, competencies and motivations of the stakeholders, have to be addressed during the 'team set up and adjustment' process. These factors should be addressed while identifying potential solutions for the challenges. In order to address the different backgrounds and tacit knowledge, KE activities related to forming teams and reviewing goals are relevant. On the other hand, in order to address the motivation of the partners, well defined roles and activities for reviewing and thus sharing the

Table 1: *A summary of the challenges for 'Team Set Up and Adjustment'*

Identified challenges	*Examples from cases*
Setting up the project, e.g., the selection of a partner (either external companies, or sites within a company)	Agreeing about IPR
	Establishing good understanding between partners about requirements
Defining the roles of different parties	Ensuring mutual benefit between partners
The optimal use of resources and competences over sites	Managing dependency to the component provider
	Documenting non-functional requirements
Dividing work, so that unnecessary dependencies over a distance can be avoided	Establishing good architecture (distribution support) and shared understanding about it
Describing the goals clearly and understandably	The identification of dependencies between partners
	Resource allocation in GSD
The communication and social skills of project members	Selecting a subcontractor

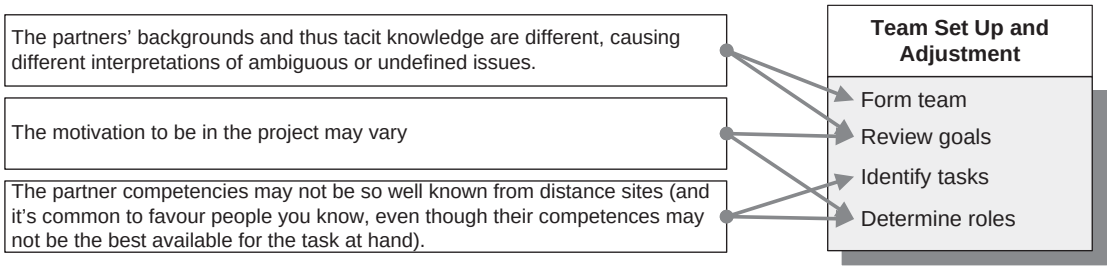


Figure 3: *Main knowledge based factors for 'Team Set Up and Adjustment'.*

Table 2: *Summary of the challenges for 'Group Problem Solving'*

Identified challenges	*Examples from cases*
Identifying problems or potential problems early	Identifying differences in requirement interpretations
The brainstorming of problems or design issues over a distance	Repeating defects found in tests effectively
The negotiation of conflicts	Resource management in GSD
Sufficient communication about design rationale and decisions	Un-communicated changes made by other partners
The availability and correct interpretation of measurement data	

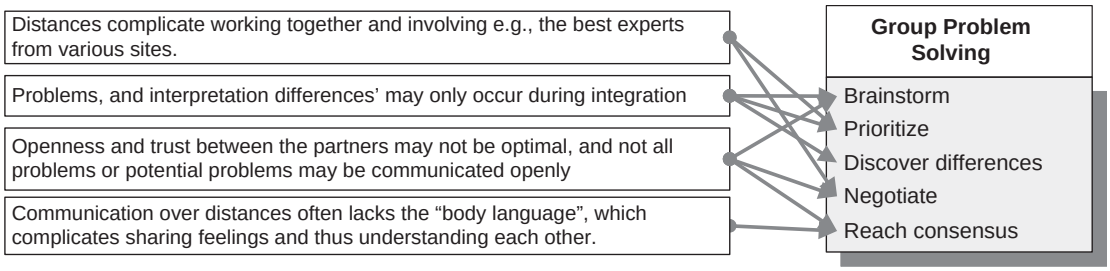


Figure 4: *Main knowledge based factors for 'Group Problem Solving'.*

knowledge of the goals of the project are useful. Relating to sharing knowledge of the competences of the project partners over sites defining roles and tasks clearly are helpful.

3.3.2. The challenges for 'Group Problem Solving' activities 'Group Problem Solving' covers activities such as brainstorming, prioritizing, discovering differences, negotiating, and reaching a consensus, for example. In the following table (Table 2), the identified knowledge intensive challenges are presented, along with examples from cases.

The main knowledge based factors causing the challenges were identified as shown in Figure 4. The relation of these factors to the activities defined by Noble is also shown in the figure.

In the 'group problem solving' process, a team engages in its 'collaborative dialog' to reach a consensus and decide what to do. If the identified factors have not been recognized and minimized, they can cause wrong conclusions and decisions. There is a great deal of tacit knowledge behind the factors and thus, solutions that increase communication, trust, openness and the awareness of each other, as well as solutions that make knowledge available in an explicit form, have to be emphasized in GSD. Activities such as prioritizing and negotiating help to address complications caused by

Table 3: *Summary of the challenges for 'Synchronize and Act'*

Identified challenges	*Examples from cases*
The follow-up and tracking of work and dependencies A good level of communication A shared understanding of the basis of the work (e.g., requirements, architecture) and dependencies Ensuring the availability of required information	The reliability of partners' schedules Creating and maintaining traceability Ensuring the shared understanding of requirements Synchronization with the integrator and component supplier Sharing test information The availability of required integration competence Subcontract management

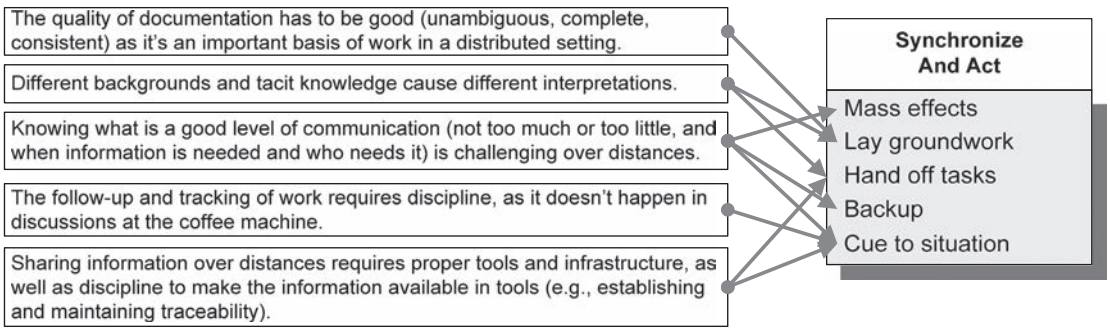


Figure 5: *Main knowledge based factors for 'Synchronize and Act'.*

distances, and interpretation differences can be addressed by brainstorming, making priorities clear and actively discovering differences. The establishing of openness and trust can be supported by activities related to negotiating, brainstorming freely and reaching a consensus that can also help in creating a better understanding of each other.

3.3.3. The challenges for 'Synchronize and Act' activities 'Synchronize and Act' covers activities such as mass effects, laying the groundwork, hand-off tasks, backups, cueing to a situation, for example. In the following table (Table 3), the identified knowledge intensive challenges are presented, along with examples from cases.

The main knowledge based factors causing the challenges were analysed and identified as shown in Figure 5. The relation of these factors to the activities defined by Noble is also shown in the figure.

In the 'synchronize and act' process, team members coordinate and help each other to achieve the most benefits from the teamwork. This coordination will fail in so far as the identified challenges and factors are not addressed. In GSD, it is important to recognize that, e.g., practices for follow-up and tracking work, knowledge sharing methods and tools, and the quality of documentation has been established. The quality of the documentation can be addressed via a proper laying of groundwork for other team members, and different interpretations can be avoided through good and coordinated hand-offs and the laying of groundwork. Sufficient communication can be ensured via backups, cueing to the situation and the massing of effects. Any challenges caused by distances can be addressed via proper hand-offs and cueing to the situation, so that the relevant information and knowledge is shared between partners.

3.3.4. The challenges for 'Individual and Shared Understanding' activities 'Individual and Shared Understanding' (~knowledge)

Table 4: *Summary of the challenges for 'Individual and Shared Understanding'*

Identified challenges	*Examples from cases*
Creating a shared understanding Knowing what individual knowledge each participant possesses Documentation is an important means to share information, and its quality is essential for success	Communication with remote peers The incompatibility of tools The openness of communication Establishing trust Ensuring sufficient knowledge to base the work and decisions on An adequate level of design Interpreting measurement data in GSD

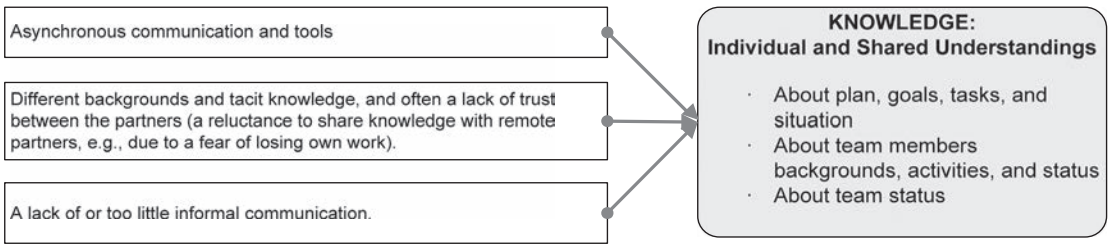


Figure 6: *Main knowledge based factors for 'Individual and Shared Understanding'.*

activities combine information and knowledge perceiving from each of the three key team activities, as well as an interactive shared understanding of team adjustments, problem settings and synchronized situations. In the following table (Table 4), the knowledge intensive challenges are presented, along with examples from cases.

The main knowledge based factors causing the challenges were analysed and identified as shown in Figure 6. The relation of these factors to the activities defined by Noble is also shown in the figure.

Individual and shared understanding, i.e., knowledge creation requires communication, communication and again communication, as communication increases mutual trust between partners. This means that informal communication is necessary and that is why selected tools should support asynchronous communication as well as the knowledge sharing process.

4. Proposed solutions

In this section, we will discuss example solutions according to the identified challenges with the perspectives proposed by Noble (2004). The presented solutions are typically things that need to be considered when carrying out a GSD project, as well as some practical way of working descriptions, which help to take into account the things mentioned. The solutions presented in this section have also usually been described in other publications, and have been chosen to be discussed here as they address the KE viewpoint well.

4.1. Solutions relating to 'Team Set Up and Adjustment'

In this section, solutions for 'Team Set Up and Adjustment' activities will be discussed. Figure 7 shows the relation of the Noble's activities to the example practical solutions explained here.

Conditions for collaboration: An organization should base its business/project decisions on a collaboration strategy, and it should understand the consequences and impact of collaboration decisions on its business (benefits and disadvantages/risks). Each partner should understand their role in the project (e.g., resource provider vs. strategic partner), as that helps in

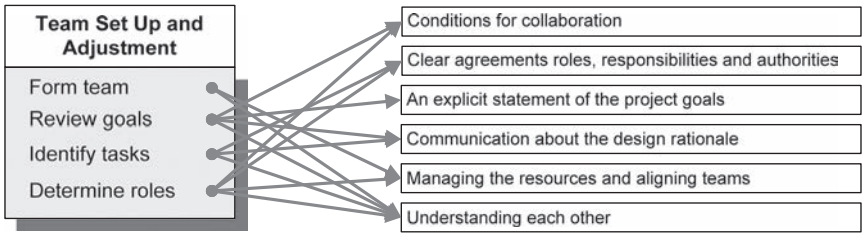


Figure 7: *Solutions relation to 'Team Set Up-up and Adjustment' activities.*

defining the required interaction and goals for the work. A rewarding policy helps to increase awareness and motivation for collaborations. Also, the creation of a functional and purposeful project organization is important for GSD project success.

Clear agreement roles, responsibilities and authorities: The roles of all of the parties involved should be clearly described and communicated, e.g., the responsibilities and authorities, including the escalation path, should be defined and communicated. Example roles include the project leader/responsible for achieving the project targets, a project management team representing the major cultures within the project, a supplier/relationship manager, team leaders and teams which are fully accountable and responsible for their results, in addition to a project level steering group including members from all of the organizations and sites.

An explicit statement of the project goals ensures that all of the project partners work on the same basis. In order to define the goals explicitly, the following aspects should be considered: the scope of the work to be performed, the risks to be incurred, the resources to be required, the tasks to be accomplished, the milestones to be tracked, the effort (cost) to be expended, and the schedules to be followed. Before a project can be planned, the objectives and scope should be established, alternative solutions should be considered and the technical and management constraints should be identified.

Communication about the design rationale, e.g., the information concerning why some design decision has been made, and why some other decision is not acceptable is important knowledge in order to avoid conflicting decisions made by other partners of the project. These are worthwhile to include in the architecture documentation, including the recommended design patterns.

Managing resources and aligning teams, in order to effectively utilize critical resources, they need to be identified, and knowledge of their availability needs to be updated and communicated continuously. In order to align the teams' work, communication (what, when, who, how), and responsibilities and dependencies within and between the teams need to be defined.

Understanding each other: In GSD, the project manager requires specific skills in addition to the usual project management and technical knowledge, namely communication skills, and knowledge of the cultures (countries, or companies), and competencies involved in the project. It is also good to analyse the different cultures who are involved in the project in the beginning, in order to become aware of the differences and thus be able to take them into account during the project.

4.2. Solutions relating to 'Group Problem Solving'

In this section, solutions for 'Group Problem Solving' activities are discussed. The relations of these solutions to the activities of the Noble model are shown in Figure 8. In addition to these solutions, some of the solutions addressed in the previous section are also valid concerning this topic, e.g., 'managing resources and aligning teams' and 'understanding each other'.

Escalation channels: Acceptance procedures and decision authorities need to be agreed upon in order to enable the management of problems

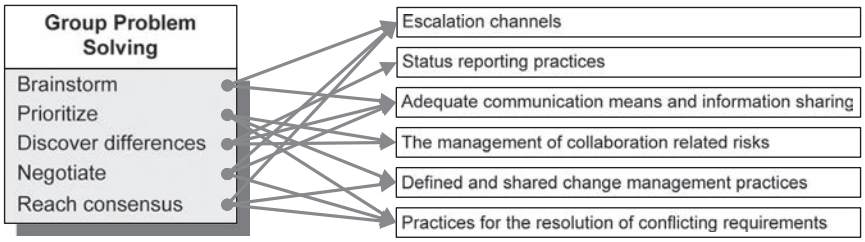


Figure 8: *Solutions relation to 'Group Problem Solving' activities.*

and conflicts. In particular, when multiple companies are involved, explicit and predefined escalation channels are required to cope with problems that cannot be controlled by the project itself, as it requires the involvement of the (authorized) management of basically all of the partners. There are two major categories of conflicts; technical, such as conflicts in design approaches and design implementations, and business, non-technical conflicts, such as schedules and task priorities.

Status reporting practices: The reporting format, reporting channels, decision authorities, and problem solving practices should be defined. The following reporting practices have been found to be useful in a distributed development:

- Distributing drafts of schedules and task assignments for each incremental release.
- Weekly task reports and meetings within a subgroup.
- Delivery reports (a description of the changes/features that are checked in).
- Quarterly sync-up meetings (the developers meet together face-to-face for a week).
- Revising all of the documents to reflect the current state of the development.
- Frequent deliveries of codes and several iteration cycles and builds.
- Frequent and incremental integration and testing.

Adequate communication means and information sharing: Adequate communication means, facilities and information sharing have a major impact in collaboration, due to the need for intensified communication. Active communication and information sharing supports the fast establishment of fluent co-operation. The communication items and roles should be defined, communication channels and tools should be defined and the availability ensured, potential communication bottlenecks should be identified and the mechanisms for managing them defined.

Managing collaboration related risks: Collaboration related risks should be managed as part of a normal risk management. It is necessary to look at the sources of technical, organizational and communication risks. Typically, risks are related to unclear assignments or specifications of work in the contract, the openness of communication, trust between the partners, and the reliability of the partner's development schedule.

Defined and shared change management practices: In GSD, the scope of the impact assessment, information sharing, and viewpoints that need to be taken into account in decision making, are affected by collaborative environment. When multiple teams or partners are involved, the levelling of change requests analysis is important to optimize the use of resources and to ensure an adequate level of communication, meaning that the changes are managed at their level of impact.

Practices for the resolution of conflicting requirements: A generic requirements interrelationship model can be used when identifying conflicts between the requirements. When the conflicting requirements have been identified, different stakeholders must decide upon which quality attributes are favoured over others, and these priorities need to be consistently respected when making decisions.

4.3. Solutions relating to 'Synchronize and Act'

In this section, solutions for 'Group Problem Solving' activities are discussed. The relations of

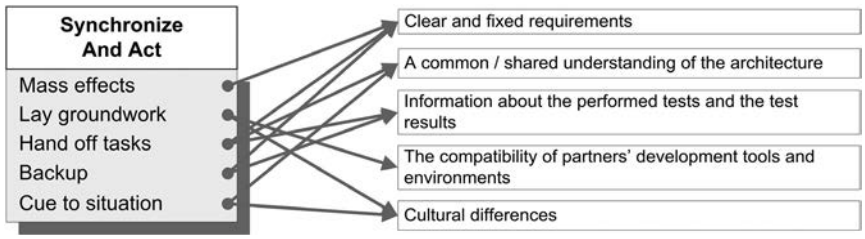


Figure 9: *Solutions relation to 'Synchronize and Act' activities.*

these solutions to the activities of the Noble model are shown in Figure 9. Some of the solutions which were addressed in the previous section are also valid relating to this topic, e.g., 'escalation channels', 'status reporting practices', 'adequate communication means and information sharing', 'managing resources and aligning teams' and 'understanding each other' help to achieve fluent co-operation during the project.

Clear and fixed requirements: The effect of unambiguous and changing requirements is higher in GSD due to the leverage effect caused by the multiple levels of control. In order to avoid misunderstandings and create a mutual vision of a project, specifications should be unambiguous and clear. It is also important to ensure that people with enough competence are involved in the requirements analysis (from all of the partners). Establishing a common understanding can be supported via continuous communication about the requirements, and by using an agreed upon structure for the requirements.

A common/shared understanding of the architecture: The establishment of a common understanding can be supported via continuous communication about the architecture, via good architecture documentation, including recommended design patterns and by architects reviewing the further work products made by other members of the project.

Information about the performed tests and test results: The responsibilities and authority for test reporting coordination and acceptance should be defined and the practices for the communication of the performed tests and test results followed. Common tools and repositories assist in the sharing of information process.

The compatibility of the partners' development tools and environments: In GSD, it is not possible to select or determine what development tools and environments each partner shall use. That is why it is important to recognize and define, for example, what kind of visibility is needed for another partner's work or how communication and data sharing can be supported between the partners during the project. This is discussed in more detail in the following section (4.4).

Cultural differences: The identification of cultural differences in a GSD project is important in order to better understand each other and to avoid problems and conflicts. Several publications exist, giving examples of differences between various cultures. It is important not to assume that the motivations, actions, and reasoning of those from other countries match yours. Failing to recognize the differences between cultures may result in some serious consequences. Note that there can also be different cultures between different companies.

4.4. *Solutions relating to 'Individual and Shared Understanding'*

The solutions presented in earlier sections are also all relevant from the 'Knowledge: Individual and Shared Understanding' viewpoint, but in this section, we will focus on one specific solution, namely the *Compatibility of partners' development tools and environments*. This is because it has become clear, through several of the cases, that if the development tools and environments are not compatible, several knowledge-related

problems have occurred. For example, if the data in different tools is not connected, whether the product meets the requirements in a collaborative development becomes untraceable, the sharing of the test environment and results is not possible if partners use different tools, and the visibility of the collaborative development status beyond the partner borders is lacking, in so far as the data is spread out between various isolated tools. These challenges have been addressed in many of the cases discussed in this paper.

In order to support the sharing of information during a GSD project, development tool interoperability and the accessibility of data are important topics to address. Over several years, we have been working on tool interoperability concepts and have developed prototypes for tool integration. These solutions aim to provide an enhanced awareness and synchronization of assets in a GSD environment, by enabling the interoperability of various software development tools in collaborative settings. The tool integration solution has been carried out in co-operation with the participating companies: the requirements have been derived from the companies and from the cases in particular. Also, the implemented solution has been validated in the industrial cases. One important aspect of our solution is that it enables the use of a company's legacy development tools, configurable for an individual partner or project needs, in order to minimize the costly and risky changes in a tool environment.

Figure 10 presents the idea behind the tool integration solution: meaning that the integration layer connects the data from different tools and provides the same view to the data for all partners. This enables the use of the same versions in different sites, as well as seeing the progress that other partners are making online.

Currently, we are also working on context aware communication support, e.g., seeing who are working on related topics online, and establishing communication with them. Also, knowledge storing, meaning the storing of relevant communication records, so that they are available for those who didn't participate in the actual communication situation, but need the information for their work, is an interesting topic. We also aim to support the creation of awareness at the workspace, i.e., finding out

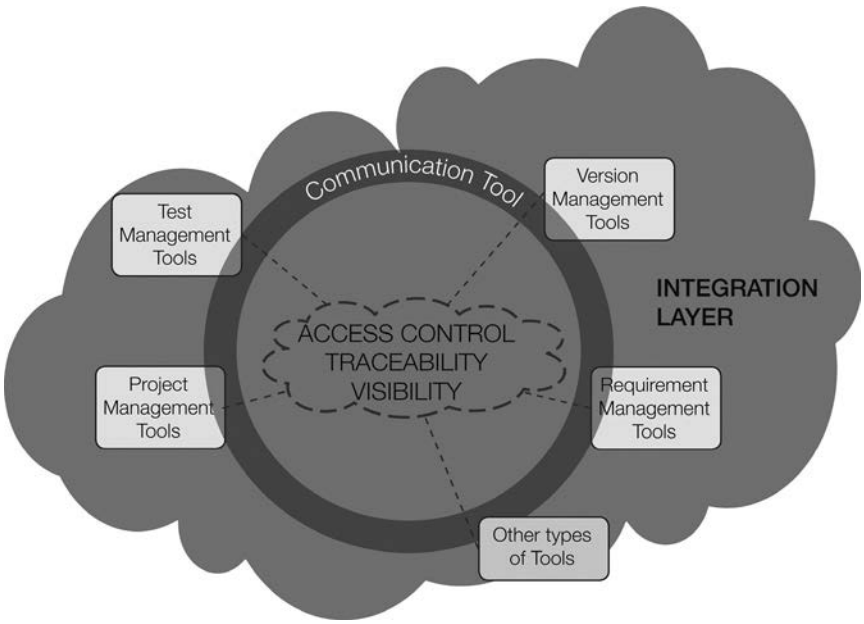


Figure 10: *Tool integration concepts.*

easily what has happened since a person was last online.

First, the complete ToolChain was developed in the Merlin project (Heinonen *et al*., 2007) and (Pesola *et al*., 2008). The main goal for the Merlin ToolChain was to evaluate and validate the concept of tool integration supporting a globally distributed development, i.e., when the work of several partners is distributed around the world, using various development tools, and needing to share information. Merlin ToolChain demonstrated that the integration of tools from different vendors is possible, and it also answered directly to the identified challenges in collaboration. Further developments of the ToolChain are reported in Eskeli & Parviainen (2010) and in the seminar presentation (Eskeli, 2010).

The experience of using the tool integration solution in industrial cases has shown that the tool integration has enabled a full transparency on real-life project data, and has provided a unified user-interface for various views (tasks, requirements, code, build & test). This has been beneficial for the projects, as it has improved the knowledge sharing while doing the same work as before, i.e., not adding extra tasks.

5. Discussion

In this article, we have introduced the challenges that companies have faced in GSD, discussed their KE aspects, and presented example solutions for addressing the challenges. In GSD, the effective and successful transfer of tacit knowledge requires extensive personal contacts, communication and trust. Cognitive perspectives have been presented as a fundamental success factor for teams in collaboration. Any knowledge gap within the team can expand into big

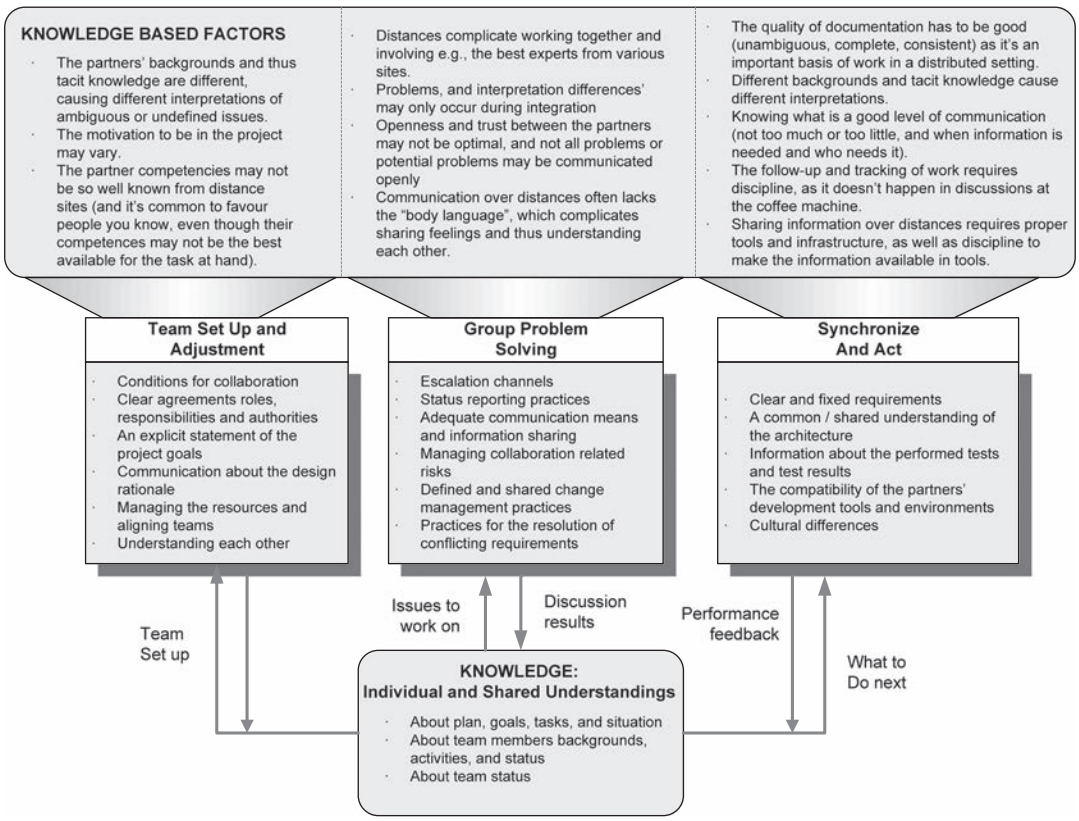


Figure 11: *Knowledge based factors relation to the team activities and solutions.*

problems and may lead to the poor sharing of information or a lack of knowledge about what to do. We used the model of Noble (2004) to emphasize the knowledge needs of distributed teams and stakeholders by analysing the challenges encountered in GSD from the KE viewpoint. This enabled us to find relevant solutions to address these challenges and to further utilize KE principles to solve GSD challenges. Figure 11 summarizes the knowledge based factors (discussed in section 3.3) and their relation to the team activities and the related solutions (discussed in section 4).

This article pointed out that a successful distributed software development requires both structured and disciplined software engineering and KM solutions. Communication management and the utilization of effective substitutes for face-to-face communication have an important role in GSD, to ensure knowledge sharing. A careful execution of project start-up activities – including the planning (dividing work, schedule, mutual deliveries), the exact definition and agreement of common rules, responsibilities, and tools used – can greatly contribute to a successful implementation. Also, ensuring the availability of information during the project to all of the parties is essential for a successful project. By understanding the nature and demands of the GSD, as well as the KE challenges in depth, software organizations will be able to reduce the risk of failure and to make their operations successful.

Acknowledgements

This paper was written within the PRISMA project (http://www.prisma-itea.org/), which is an ITEA 2 project, number 07024. The authors would like to thank the support of ITEA (http://www.itea2.org/) and Tekes – the Finnish Funding Agency for Technology and Innovation (http://www.tekes.fi/eng/).

References

Aranda, G.N., A. Vizcaino, A. Cechich and M. Piattini (2006) Technology selection to improve global collaboration, in *Proceedings of International Conference on Global Software Engineering ICGSE '06*, Florianopolis, Brazil, pp. 223–232.

Bhat, J.M., G. Mayank and S.N. Murthy (2006) Overcoming requirements engineering challenges: lessons from offshore outsourcing, *Journal of IEEE Software, IEEE Computer Society*, **23**, 38–44.

Carmel, E. and R. Agarwal (2001) Tactical approaches for alleviating distance in global software development, *Journal of IEEE Software, IEEE Computer Society*, **18**, 22–29.

CHAOS Reports (1996, 1998, 2000, 2002, 2004 and 2006) the Standish Group International Inc. Available at http://www.standishgroup.com (accessed 10 January 2011)

CMMI for development, version 1.2. (2006) Technical Report CMU/SEI-2006-TR-008. Available at http://www.sei.cmu.edu/cmmi/ (accessed 10 January 2011)

Damian, D. and D. Moitra (2006) Global software development: How far have we come?, *Journal of IEEE Software*, **23**, 17–19.

Damian, D.E. and D. Zowghi (2002) The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization, in *Proceeding of IEEE Joint International Conference on Requirements Engineering*. pp. 319–328.

Davenport, T.H. and L. Prusak (1998) *Working Knowledge*, Boston, USA: Harvard Business School Press.

De Souza, C.R.B., S.D. Basaveswara and D.F. Redmiles (2002) Supporting Global Software Development with Event Notification Servers, in Proceedings of Global Software Development, Workshop #9, organized in the International Conference on Software Engineering (ICSE) 2002, Orlando, Florida, USA.

Desouza, K.C., Y. Awazu and P. Baloh (2006) Managing knowledge in global software development efforts: issues and practices, *Journal of IEEE Software*, **23**, 30–37.

Eskeli, J. (2010) Tools for breaking the walls, SameRoomSpirit seminar presentation. Available at: http://conference.erve.vtt.fi/srs2010/files/EskeliJuho_Tools_for_breaking_the_walls_20100506.pdf (accessed 10 January 2011)

Eskeli, J. and P. Parviainen (2010) Supporting Hardware-related Software Development with Integration of Development Tools, in *Proceedings of Fifth International Conference on Software Engineering Advances ICSEA'10*, August 22–27, 2010, Nice, France, pp. 353–358.

Gotel, O. and A. Finkelstein (1994) An Analysis of the Requirements Traceability Problem, in *Proceedings of the 1st International Conference on Requirements Engineering*, April 18–22, 1994, pp. 94–101.

Heinonen, S., J. Kääriäinen and J. Takalo (2007) Challenges in Collaboration: Tool Chain Enables

Transparency Beyond Partner Borders, in *Proceedings of 3rd International Conference Interoperability for Enterprise Software and Applications 2007*, Funchal, Portugal.

Herbsleb, J.D. (2007) Global Software Engineering: the future of socio-technical coordination, in *Proceedings of Future of Software Engineering FOSE '07*, May 23–25, 2007 IEEE Computer Society.

Herbsleb, J.D., A. Mockus, T.A. Finholt and R.E. Grinter (2001) An Empirical Study of Global Software Development: Distance and Speed, in *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, May 12–19, 2001, Toronto, Ontario, Canada, pp. 81–90.

Herbsleb, J.D. and D. Moitra (2001) Global software development, *Journal of IEEE Software*, **18**, 16–20.

Hyysalo, J., P. Parviainen and M. Tihinen (2006) Collaborative Embedded Systems Development: Survey of State of the Practice, in *Proceedings of ECBS'06*, 13th Annual IEEE International Conference on the Engineering of Computer Based Systems, March 27–30, Germany 2006.

Jimènez, M., M. Piattini and A. Vizcaíno (2009) Challenges and Improvements in Distributed Software Development: A Systematic Review. Hindawi Publishing Corporation, Advances in Software Engineering Volume 2009, Article ID 710971, 14pp.

Komi-Sirviö, S. and M. Tihinen (2005) Lessons learned by participants of distributed software development, *Journal of Knowledge and Process Management*, **12**, 108–122.

Kommeren, R. and P. Parviainen (2007) Philips experiences in global distributed software development, *Journal of Empirical Software Engineering*, **12**, 647–660.

Layman, L., L. Williams, D. Damian and H. Bures (2006) Essential communication practices for Extreme Programming in a global software development team. Information and Software Technology Volume 48, Issue 9, September 2006, Special Issue Section: Distributed Software Development, pp. 781–794.

MERLIN (2004-2007) ITEA project, Embedded Systems Engineering in Collaboration, Available at http://virtual.vtt.fi/virtual/proj1/projects/merlin/index.html (accessed 10 January 2011)

MERLIN Collaboration Handbook (2007) Available at http://www.merlinhandbook.org (accessed 10 January 2011)

Noble, D. (2002) A Cognitive Description of Collaboration and Coordination to Help Teams Identify and Fix Problems, in *Proceedings of 7th International Command and Research Control and Technology Symposium*, September 16–20, Quebec, Canada.

Noble, D. (2004) Knowledge Foundations of Effective Collaboration, in *Proceedings of 9th International Command and Control Research and Technology Symposium*, September 14–16, Copenhagen, Denmark.

Noll, J., S. Beecham and I. Richardson (2010) Global software development and collaboration: barriers and solutions, *Journal of ACM Inroads*, **1**, 66–78.

Nonaka, I. (1994) A dynamic theory of organisational knowledge creation, *Organisation Science*, **5**, 14–37.

Olson, G.M. and J.S. Olson (2000) Distance matters, *Human-Computer Interaction*, **15**, 139–178.

Parviainen, P., J. Eskeli, T. Kynkäänniemi and M. Tihinen (2008) Merlin Collaboration Handbook: Challenges and Solutions, in *Proceedings of the 3rd International Conference on Software and Data Technologies ICSOFT 2008*. Porto, Portugal, 5–8 July 2008. INSTICC. Vol. SE (2008), No: GSDCA/M/, 339–346.

Pesola, J-P., J. Eskeli, P. Parviainen, R. Kommeren and M. Gramza (2008) Experiences of tool integration: development and validation, in *Enterprise Interoperability III – New Challenges and Industrial Approaches*, K. Mertins, R. Ruggaber, K. Popplewell and X. Xu (eds), London, UK: Springer, 499–510.

PRISMA (2009-2011) ITEA2 project, Productivity in Collaborative Systems Development, Available at: http://www.prisma-itea.org (Accessed 10 January 2011)

Rus, I. and M. Lindvall (2002) Knowledge management in software engineering, *IEEE Journal of Software*, **19**, 26–38.

Sommerville, I. and P. Sawyer (1997) *Requirements Engineering: A Good Practice Guide*, Chichester: John Wiley & Sons.

Van Solingen, R., P. Parviainen and M. Tihinen (2008) Solutions for challenges in global collaborative product development, ITEA Innovation report, March 2008, available at http://www.itea2.org/attachments/428/innovation_report_MERLIN.pdf (accessed 10 January 2011)

VTT (2011) VTT Technical Research Centre of Finland, available at http://www.vtt.fi/?lang=en (accessed 10 January 2011)

Yin, R.K. (2003) *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series*, Vol. 5, USA, Sage Publications Inc., 181pp.

The authors

Päivi Parviainen

Päivi Parviainen is a Principal Scientist and team manager in the Software Technologies center at VTT Technical Research Centre of Finland. She has worked at VTT since 1995. She has

experience in software process improvement, measurement, software reuse, software development tools and their integration, systems and software requirements engineering and global software development practices, for example.

Maarit Tihinen

Maarit Tihinen is a Research Scientist and Quality Manager at VTT (VTT Technical Research Centre of Finland). Before joining VTT at year 2000, she worked as a mathematics and information technology teacher at Kemi-Tornio Polytechnic during the nineties. Her research interests are focused on software processes, especially, on improving software processes as well as measurements and metrics.

PAPER V

Requirements engineering

Dealing with the complexity of sociotechnical systems development

In: Requirements Engineering for Sociotechnical Systems. Chapter 1, pp. 1–20.

Chapter I

Requirements Engineering: Dealing with the Complexity of Sociotechnical Systems Development

Päivi Parviainen,
VTT Technical Research Centre of Finland, VTT Electronics, Finland

Maarit Tihinen,
VTT Technical Research Centre of Finland, VTT Electronics, Finland

Marco Lormans, Delft University of Technology, The Netherlands

Rini van Solingen,
LogicaCMG Technical Software Engineering (RTSE1), The Netherlands

Abstract

This chapter introduces requirements engineering for sociotechnical systems. Requirements engineering for sociotechnical systems is a complex process that considers product demands from a vast number of viewpoints, roles, responsibilities, and objectives. This chapter explains the requirements engineering terminology and describes the requirements engineering process in detail, with examples of available methods for the main process activities. The main activities described include system requirements development, requirements allocation and flow-down, software

requirements development, and continuous activities, including requirements documentation, requirements validation and verification, and requirements management. As requirements engineering is the process with the largest impact on the end product, it is recommended to invest more effort in both industrial application as well as research to increase understanding and deployment of the concepts presented in this chapter.

Introduction[1]

The concept of sociotechnical systems was established to stress the reciprocal interrelationship between humans and machines and to foster the program of shaping both theoretical and social conditions of work (Ropohl, 1999). A sociotechnical system can be regarded as a theoretical construct for describing and explaining technology generally. This chapter helps to describe a multidisciplinary role of requirements engineering as well as the concept of workflow and patterns for social interaction within the sociotechnical systems research area.

Requirements engineering is generally accepted as the most critical and complex process within the development of sociotechnical systems (Juristo, Moreno, & Silva, 2002; Komi-Sirviö & Tihinen, 2003; Siddiqi, 1996). The main reason is that the requirements engineering process has the most dominant impact on the capabilities of the resulting product. Furthermore requirements engineering is *the process* in which the most diverse set of product demands from the most diverse set of stakeholders is being considered. These two reasons make requirements engineering complex as well as critical.

This chapter first introduces background information related to requirements engineering, including the terminology used and the requirements engineering process in general. Next a detailed description of the requirements engineering process, including the main phases and activities within these phases, is presented. Each phase will be discussed in detail, with examples of useful methods and techniques.

Background

A requirement is a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents (IEEE Std 610.12, 1990). A well-formed requirement is a statement of system functionality (a capability) that must be met or possessed by a system to satisfy a customer's need or to achieve a customer's objective, and that is qualified by measurable conditions and bounded by constraints (IEEE Std 1233, 1998).

Requirements are commonly classified as (IEEE Std 830, 1998):

- *Functional:* A requirement that specifies an action that a system must be able to perform, without considering physical constraints; a requirement that specifies input/output behavior of a system.
- *Non-functional:* A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. Non-functional requirements are often classified into the following categories:
- *Performance requirements:* A requirement that specifies performance characteristics that a system or system component must possess, for example, max. CPU-usage, max. memory footprint.
- *External interface requirements:* A requirement that specifies hardware, software, or database elements with which a system or system component must interface or that sets forth constraints on formats, timing, or other factors caused by such an interface.
- *Design constraints:* A requirement that affects or constrains the design of a system or system component, for example, language requirements, physical hardware requirements, software development standards, and software quality assurance standards.
- *Quality attributes:* A requirement that specifies the degree to which a system possesses attributes that affect quality, for example, correctness, reliability, maintainability, portability.

Requirements engineering contains a set of activities for discovering, analyzing, documenting, validating, and maintaining a set of requirements for a system (Sommerville & Sawyer, 1997). Requirements engineering is divided into two main groups of activities, requirements development and requirements management. *Requirement development* includes activities related to discovering, analyzing, documenting, and validating requirements, where as *requirement management* includes activities related to maintenance, namely identification, traceability, and change management of requirements.

Requirements validation consists of activities that try to confirm that the behaviour of a developed system meets its *user needs*. Requirements verification consists of those activities that try to confirm that the product of a system development process meets its technical specifications (Stevens, Brook, Jackson, & Arnold, 1998). Verification and validation include:

- Defining the verification and validation requirements, that is, principles on how the system will be tested.
- Planning the verification and validation.
- Capturing the verification and validation criteria (during requirements definition).

- Planning of test methods and tools.
- Planning and conducting reviews.
- Implementing and performing the tests and managing the results.
- Maintaining traceability.
- Auditing.

In sociotechnical systems software is understood as a part of the final product. System requirements are captured to identify the functioning of the system, from which software requirements are derived. Deciding which functionality is implemented where, and by which means (software, hardware, mechanics, and so forth) is merely a technical decision process in which feasibility, dependability, and economics play a role. A well-structured and technically sound requirements engineering process is, therefore, of utmost importance.

Requirements Engineering Process

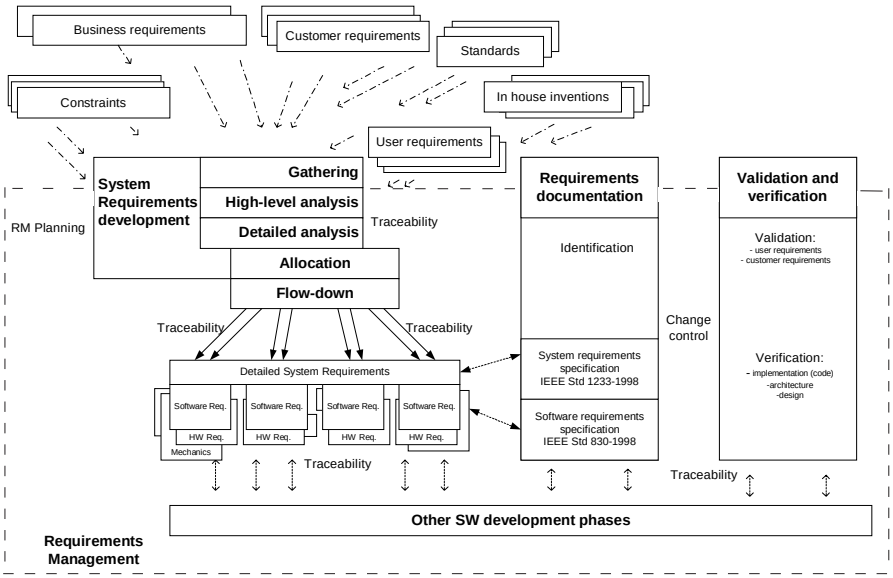
Figure 1 describes a requirements engineering process where the main processes of system and software requirements engineering are depicted. Requirements engineering activities cover the entire system and software development lifecycle. On the other hand the requirements engineering process is iterative and will go into more detail in each iteration. In addition the figure indicates how requirements management (RM) is understood as a part of the requirements engineering process. The process is based on Kotonya and Sommerville (1998), Sailor (1990), Thayer and Royce (1990).

The process describes requirements engineering for sociotechnical systems, where software requirements engineering is a part of the process. Traditionally requirements engineering is performed in the beginning of the system development lifecycle (Royce, 1970). However, in large and complex systems development, developing an accurate set of requirements that would remain stable throughout the months or years of development has been realized to be impossible in practice (Dorfman, 1990). Therefore requirements engineering is an incremental and iterative process, performed in parallel with other system development activities such as design.

The main high-level activities included in the requirements engineering process are:

1) *System requirements development,* including requirements gathering/elicitation from various sources (Figure 1 shows the different sources for requirements), requirements analysis, negotiation, priorisation and agreement of raw requirements, and system requirements documentation and validation.
2) *Requirements allocation and flow-down,* including allocating the captured requirements to system components and defining, documenting, and validating detailed system requirements.

Figure 1. System and software requirements engineering (Parviainen, Hulkko, Kääriäinen, Takalo, & Tihinen, 2003)



3) *Software requirements development,* including analyzing, modeling and validating both the functional and quality aspects of a software system, and defining, documenting, and validating the contents of software subsystems.

4) *Continuous activities,* including requirements documentation, requirements validation and verification, and requirements management.

Each of these high-level activities will be further detailed in the following sections.

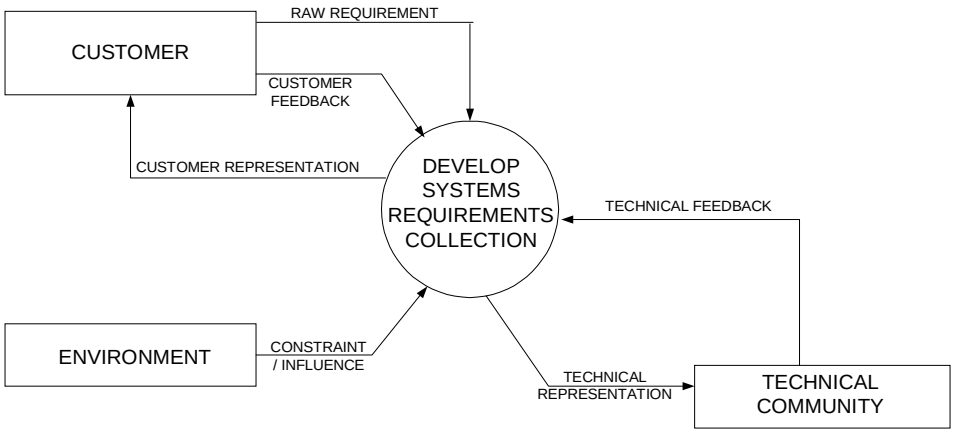
System Requirements Development

The main purpose of the system requirements development phase is to examine and gather desired objectives for the system from different viewpoints (for example, customer, users, system's operating environment, trade, and marketing). These objectives are identified as a set of functional and non-functional requirements of the system. Figure 2 shows the context for developing system requirements specification (SyRS).

1. Requirements Gathering/Elicitation from Various Sources

Requirements gathering starts with identifying the stakeholders of the system and collecting (that is, eliciting) raw requirements. Raw requirements are requirements that

Figure 2. Context for developing SyRS (IEEE Std 1233, 1998)



have not been analyzed and have not yet been written down in a well-formed requirement notation. Business requirements, customer requirements, user requirements, constraints, in-house ideas and standards are the different viewpoints to cover. Typically specifying system requirements starts with observing and interviewing people (Ambler, 1998). This is not a straightforward task, because users may not possess the overview on feasibilities and opportunities for automated support. Furthermore user requirements are often misunderstood because the requirements collector misinterprets the users' words. In addition to gathering requirements from users, standards and constraints (for example, the legacy systems) also play an important role in systems development.

2. Requirements Analysis and Documentation

After the raw requirements from stakeholders are gathered, they need to be analyzed within the context of business requirements (management perspective) such as cost-effectiveness, organizational, and political requirements. Also, the requirements relations, that is, dependencies, conflicts, overlaps, omissions, and inconsistencies, need to be examined and documented. Finally the environment of the system, such as external systems and technical constraints, need to be examined and explicated.

The gathering of requirements often reveals a large set of raw requirements that, due to cost and time constraints, cannot entirely be implemented in the system. Also the identified raw requirements may be conflicting. Therefore, negotiation, agreement, communication, and priorisation of the raw requirements are also an important part of the requirements analysis process.

The analyzed requirements need to be documented to enable communication with stakeholders and future maintenance of the requirements and the system. Requirements documentation also includes describing the relations between requirements. During requirements analysis it gives added value to record the rationale behind the decisions made to ease future change management and decision making.

Table 1. System requirements development methods

Activity	*Example methods*	*Description*
Gathering requirements	*Ethnography* (Suchman, 1983) *Protocol Analysis* (Ericsson & Simon, 1993)	*Observing methods* use techniques that may help to understand the thoughts and needs of the users, even when they cannot describe these needs or they do not exactly know what they want.
	Focus groups (Maguire, 1998) *JAD* (Joint Application Development) (Ambler, 1998)	*Meeting techniques* cover separate techniques for meetings and workshops for gathering and developing requirements from different stakeholders.
	Volere (Robertson & Robertson, 1999)	Provides a generic process for gathering requirements, ways to elicit them from users, as well as a process for verifying them.
Requirements analysis	*QFD* (Quality Function Deployment) (Revelle, Moran, Cox, & Moran, 1998)	Identifying customer needs, expectations, and requirements, and linking them into the company's products.
	SCRAM (Scenario-based Requirements Engineering) (Sutcliffe, 1998)	Develop requirements (whole RE) using scenarios. The scenarios are created to represent paths of possible behavior through use cases, and these are then investigated to develop requirements.
	SSADM (Structured System Analysis and Design Methodology) (Ashworth & Goodland, 1990)	Can be used in the analysis and design stages of systems development. It specifies in advance the modules, stages, and tasks that have to be carried out, the deliverables to be produced, and the techniques used to produce those deliverables.
Negotiation and priorisation	*CORE* (Controlled Requirements Expression) (Mullery, 1979) *WinWin* approach (Bose, 1995)	The purpose of *viewpoint-oriented methods* is to produce or analyze requirements from multiple viewpoints. They can be used while resolving conflicts or documenting system and software requirements.
System requirements documentation	*IEEE Std 1233-1998*	*Standards* define the contents of a SyRS.
	VDM (Vienna Development Model) (Björner & Jones, 1978) *Specification language Z* (Sheppard, 1995)	In *formal methods*, requirements are written in a statement language or in a formal -- mathematical -- way.
	HPM (Hatley-Pirbhai Methodology) (Hatley & Pirbhai, 1987)	Gives support for documenting and managing of system requirements.
	VORD (Viewpoint-Oriented Requirements Definition) (Kotonya & Sommerville, 1996)	Helps to identify and prioritize requirements and also can be utilized when documenting system and software requirements.

3. System Requirements Validation and Verification

In system requirements development, validation and verification activities include validating the system requirements against raw requirements and verifying the correctness of system requirements documentation. Common techniques for validating requirements are requirements reviews with the stakeholders and prototyping.

Table 1 contains examples of requirements engineering methods and techniques used during the system requirements development phase. The detailed descriptions of the methods have been published in Parviainen et al. (2003).

Several methods for gathering, eliciting, and analyzing requirements from users and other stakeholders can be used. Table 1 includes several observing methods (for example,

ethnography), meeting techniques (for example, focus groups) and analyzing techniques (for example, QFD) that can be used to gather requirements and avoid misunderstandings of users needs. The methods help in identifying needs of individuals and converting them into requirements of a desired product. At the same time social actions and workflows, safety-critical aspects, or technical constraints have to be taken into consideration. The results of the system requirements development phase are captured as top-level system requirements that are used as input for the allocation and flow-down phase.

Allocation and Flow-Down

The requirements allocation and flow-down process' purpose is to make sure that all system requirements are fulfilled by a subsystem or by a set of subsystems collaborating together. Top-level system requirements need to be organized hierarchically, helping to view and manage information at different levels of abstraction. The requirements are decomposed down to the level at which the requirement can be designed and tested; thus, allocation and flow-down may be done for several hierarchy levels. The level of detail increases as the work proceeds down in the hierarchy. That is, system-level requirements are general in nature, while requirements at low levels in the hierarchy are very specific (Leffingwell & Widrig, 2000; Stevens et al., 1998).

The top-level system requirements defined in the system requirements development phase (see previous subsection) are the main input for the requirements allocation and flow-down phase. In practice, system requirements development and allocation and flow-down are iterating; as the system level requirements are being developed, the elements that should be defined in the hierarchy should also be considered. By the time a draft of the system requirements is complete, a tentative definition of at least one and possibly two levels of system hierarchy should be available (Dorfman, 1990).

1. Requirements Allocation

Allocation is architectural work carried out in order to design the structure of the system and to issue the top-level system requirements to subsystems. Architectural models provide the context for defining how applications and subsystems interact with one another to meet the requirements of the system. The goal of architectural modeling, also commonly referred to as high-level modeling or global modeling, is to define a robust framework within which applications and component subsystems may be developed (Ambler, 1998).

Each system level requirement is allocated to one or more elements at the next level (that is, it is determined which elements will participate in meeting the requirement). Allocation also includes allocating the non-functional requirements to system elements. Each system element will need an apportionment of the non-functional requirements (for example, performance requirement). However, not all requirements are allocable; non-allocable requirements are items such as environments, operational life, and design standards, which apply unchanged across all elements of the system or its segments. The

allocation process is iterative; in performing the allocation, needs to change the system requirements (additions, deletions, and corrections) and/or the definitions of the elements can be found (Dorfman, 1990; Nelsen, 1990; Pressman, 1992; Sailor, 1990).

The overall process of the evaluation of alternative system configurations (allocations) includes:

- Definition of alternative approaches.
- Evaluation of alternatives.
- Selection of evaluation criteria; performance, effectiveness, life-cycle cost factors.
- Application of analytical techniques (for example, models).
- Data generation.
- Evaluation of results.
- Sensitivity analysis.
- Definition of risk and uncertainty.
- Selection of the configuration (Blanchard & Fabrycky, 1981; Pressman, 1992).

Once the functionality and the non-functional requirements of the system have been allocated, the system engineer can create a model that represents the interrelationship between system elements and sets a foundation for later requirements analysis and design steps. The decomposition is done right when:

- Distribution and partitioning of functionality are optimized to achieve the overall functionality of the system with minimal costs and maximum flexibility.
- Each subsystem can be defined, designed, and built by a small or at least modest-sized team.
- Each subsystem can be manufactured within the physical constraints and technologies of the available manufacturing processes.
- Each subsystem can be reliably tested as a subsystem subject to the availability of suitable fixtures and harnesses that simulate the interfaces to the other system.
- Appropriate regard is given to the physical domain – the size, weight, location, and distribution of the subsystems – that has been optimized in the overall system context (Leffingwell & Widrig, 2000).

2. Requirements Flow-Down

Flow-down consists of writing requirements for the lower-level elements in response to the allocation. When a system requirement is allocated to a subsystem, the subsystem must have at least one requirement that responds to the allocation. Usually more than one requirement will be written. The lower-level requirement(s) may closely resemble the higher-level one or may be very different if the system engineers recognize a capability

that the lower-level element must have to meet the higher-level requirements. In the latter case, the lower-level requirements are often referred to as "derived" (Dorfman, 1990).

Derived requirements are requirements that must be imposed on the subsystem(s). These requirements are derived from the system decomposition process. As such alternative decompositions would have created alternative derived requirements. Typically there are two subclasses of derived requirements:

- Subsystem requirements that must be imposed on the subsystems themselves but do not necessarily provide a direct benefit to the end user.
- Interface requirements that arise when the subsystems need to communicate with one another to accomplish an overall result. They will need to share data or power or a useful computing algorithm (Leffingwell & Widrig, 2000).

In the allocation and flow-down phase, requirements identification and traceability have to be ensured both to higher-level requirements as well as between requirements on the same level. Also the rationale behind design decisions should be recorded in order to ensure that there is enough information for verification and validation of the next phases' work products and change management.

The flowing down of the top-level system requirements through the lower levels of the hierarchy until the hardware and software component levels are reached in theory produces a system in which all elements are completely balanced, or "optimized." In the real world, complete balance is seldom achieved due to fiscal, schedule, and technological constraints (Sailor, 1990; Nelsen, 1990).

Table 2 includes few examples of methods available for the allocation and flow-down.

The results of allocation and flow-down are detailed system-level requirements and the "architectural design" or "top-level design" of the system. Again needs to change the system requirements (additions, deletions, and corrections) and/or the definitions of the

Table 2. Allocation and flow-down methods

Activity	*Example methods*	*Description*
Allocation	*SRA* (System Requirements Allocation Methodology) (Hadel & Lakey, 1995)	A customer-oriented systems engineering approach for allocating top-level quantitative system requirements. It aims at creating optimized design alternatives, which correspond to the customer requirements using measurable parameters.
	ATAM (Architecture Trade-off Analysis Method) (Kazman, Klein, Barbacci, Longstaff, Lipson, & Carriere, 1998)	Helps for performing trade-off analysis and managing non-functional requirements during allocation.
	HPM (Hatley-Pirbhai Methodology) (Hatley & Pirbhai, 1987)	Verifies requirements allocation and manages changes during allocation phase.
	QADA (Matinlassi & Niemelä, 2002) *FAST* (Weiss & Lai, 1999)	Methods for architecture design and analysis. See more from Dobrica & Niemelä, 2002.
Flow-down	*ATAM* (Architecture Trade-off Analysis Method) (Kazman et al., 1998) *HPM* (Hatley & Pirbhai, 1987)	Facilitates communication between stakeholders for gaining a rationale of requirements flow-down.

system elements may be found. These are then fed back to the system requirements development process. Allocation and flow-down starts as a multi-disciplinary activity, that is, subsystems may contain hardware, software, and mechanics. Initially they are considered as one subsystem; in later iterations the different disciplines are considered separately. Software requirements development will be described in detail in the next section.

Software Requirements Development

The software requirements development process is the activity determining which functionality of the system will be performed by software. Documenting this functionality together with the non-functional requirements in a software requirements specification is part of this phase. Through the system mechanism of flow-down, allocation, and derivation, a software requirements specification will be established for each software subsystem, software configuration item, or component (Thayer & Royce, 1990).

1. Software Requirements Analysis

Software requirements analysis is a software engineering task that bridges the gap between system-level software allocation and software design. Requirements analysis enables the specification of software functions and performance, an indication of the software interfaces with other system elements, and the establishment of design constraints that the software must meet. Requirements analysis also refines the software allocation and builds models of the process, data, and behavioral domains that will be treated by software. Prioritizing the software requirements is also part of software requirements analysis. To support requirements analysis, the software system may be modelled, covering both functional and quality aspects.

2. Software Requirements Documentation

In order to be able to communicate software requirements and to make changes to them, they need to be documented in a software requirements specification (SRS). An SRS contains a complete description of the external behavior of the software system. It is possible to complete the entire requirements analysis before starting to write the SRS. However it is more likely that as the analysis process yields aspects of the problem that are well understood, the corresponding section of the SRS is written.

3. Software Requirements Validation and Verification

Software requirements need to be validated against system-level requirements, and the SRS needs to be verified. Verification of SRS includes, for example, correctness, consistency, unambiguousness, and understandability (IEEE Std 830, 1998).

A requirements traceability mechanism to generate an audit trail between the software requirements and final tested code should be established. Traceability should be maintained to system-level requirements, between software requirements, and to later phases, for example, architectural work products.

Table 3. Software requirements development methods

Activity	*Example methods*	*Description*
Software requirements analysis	*OMT* (Object Modeling Technique) (Bourdeau & Cheng, 1995) *Shlaer-Mellor Object-Oriented Analysis Method* (Shlaer & Mellor, 1988) *UML* (Unified Modeling Language) (Booch, Jacobson, & Rumbaugh, 1998)	*Object-oriented methods* model systems in an object-oriented way or support object-oriented development in the analysis and design phases.
	SADT (Structured Analysis and Design Technique) (Schoman & Ross, 1977) *SASS* (Structured Analysis and System Specification) (Davis, 1990)	*Structured methods* analyze systems from process and data perspective by structuring a project into small, well-defined activities and specifying the sequence and interaction of these activities. Typically diagrammatic and other modeling techniques are used during analysis work.
	B-methods (Schneider, 2001) *Petri Nets* (Girault & Valk, 2002; Petri, 1962)	Formal methods are often used for safety-critical systems.
	XP (Extreme Programming) (Beck, 1999)	Agile methods are not specially focused on RE, but they have an integral point of view where RE is one of the activities of the whole cycle. See more from Abrahamsson et al., 2002.
	CARE (COTS-Aware Requirements Engineering) (Chung, Cooper, & Huynh,, 2001) *OTSO* (Off-the-Shelf Option) (Kontio, 1995)	Specific methods for RE when using COTS (Commercial off-the-shelf). COTS is a ready-made software product that is supplied by a vendor and has specific functionality as part of a system.
	Planguage (Gilb, 2003)	Consists of a software systems engineering language for communicating systems engineering and management specifications, and a set of methods providing advice on best practices.
Requirements documentation	*IEEE Std 830-1998*	IEEE defines contents of an SRS. The standard doesn't describe sequential steps to be followed but defines the characteristics of a good SRS and provides a structure template for the SRS. This template can be used in documenting the requirements and also as a checklist in other phases of the requirements engineering process.
Requirements validation	*Volere* (Robertson & Robertson, 1999)	Provides process for gathering/eliciting and validating both system and software requirements.
	Storyboard Prototyping (Andriole, 1989)	Sequences of computer-generated displays, called storyboards, are used to simulate the functions of the formally implemented system beforehand. This supports the communication of system functions to the user and makes the trade-offs non-functional and functional requirements visible, traceable and analyzable.
	Also several other methods, such as object-oriented methods, provide some support for validation and verification	

The outcome of the software requirements development phase is a formal document including a baseline of the agreed software requirements. According to SPICE (1998), as a result of successful implementation of the process:

- The requirements allocated to software components of the system and their interfaces will be defined to match the customer's stated needs.
- Analyzed, correct, and testable software requirements will be developed.
- The impact of software requirements on the operating environment will be understood.
- A software release strategy will be developed that defines the priority for implementing software requirements.
- The software requirements will be approved and updated as needed.
- Consistency will be established between software requirements and software designs.
- The software requirements will be communicated to affected parties.

Table 3 gives examples of methods or techniques available for software requirements development.

Continuous Activities

Documentation, validation, and verification of the continuous activities are included in the main process phase where the activity is mentioned the first time. Only requirements management viewpoints (identification, traceability, and change management) are discussed in this section.

Requirements management controls and tracks changes of agreed requirements, relationships between requirements, and dependencies between the requirements documents and other documents produced during the systems and software engineering process (Kotonya & Sommerville, 1998). Requirements management is a continuous and cross-section process that begins from requirements management planning and continues via activities of identification, traceability, and change control during and after requirements development process phases. Requirements management continues also after development during maintenance, because the requirements continue to change (Kotonya & Sommerville, 1998; Lauesen, 2002). Each of the requirements management activities is introduced in the following.

1. Requirements Identification

Requirements identification practices focus on the assignment of a unique identifier for each requirement (Sommerville & Sawyer, 1997). These unique identifiers are used to refer

to requirements during product development and management. Requirements' identification support can be divided into the three classes (Berlack, 1992; Sommerville & Sawyer, 1997):

1. Basic numbering systems
   - Significant numbering
   - Non-significant numbering
2. Identification schemes
   - Tagging
   - Structure-based identification
   - Symbolic identification
3. Techniques to support and automate the management of items
   - Dynamic renumbering
   - Database record identification
   - Baselining requirements

2. Requirements Traceability

Requirements traceability refers to the ability to describe and follow the life of a requirement and its relations with other development artefacts in both forward and backward direction (Gotel, 1995). This is especially important for trade-off analysis, impact analysis, and verification and validation activities. If traceability is not present, it is very difficult to identify what the effects of proposed changes are and whether accepted changes are indeed taken care of.

3. Requirements Change Management

Requirements change management refers to the ability to manage changes to requirements throughout the development lifecycle. Change management, in general, includes identifying, analyzing, deciding on whether a change will be implemented, implementing and validating change requests. Change management is sometimes said to be the most complex requirements engineering process (Hull, Jackson, & Dick, 2002). A change can have a large impact on the system, and estimating this impact is very hard. Requirements traceability helps making this impact explicit by using downward and upward traceability. For every change, the costs and redevelopment work have to be considered before approving the change. Change management has a strong relationship with baselining. After requirements' baselining, changes to the requirements need to be incorporated by using change control procedures (Hooks & Farry, 2001). Examples of requirements management methods, techniques, or approaches have been listed in Table 4.

Requirements management tools have been developed because of the problems of managing unstable requirements and the large amount of data collected during requirements engineering process. A large set of tools – both commercial and non-commercial) – is available; for examples, see Parviainen et al, 2003. Requirements management tools collect together the system requirements in a database or repository and provide a range of facilities to access the information about the requirements (Kotonya & Sommerville, 1998). According to Lang & Duggan (2001), software requirements management tools must be able to:

- Maintain unique identifiable description of all requirements.
- Classify requirements into logical user-defined groups.
- Specify requirements with textual, graphical, and model-based description.
- Define traceable associations between requirements.
- Verify the assignments of user requirements to technical design specifications.
- Maintain an audit trail of changes, archive baseline versions, and engage a mechanism to authenticate and approve change requests.
- Support secure, concurrent co-operative work between members of a multidisciplinary development team.
- Support standard systems modeling techniques and notations.
- Maintain a comprehensive data dictionary of all project components and requirements in a shared repository.

Table 4. Requirements management methods

Activity	*Example methods*	*Description*
Requirements traceability	Cross reference, traceability matrices, automated traceability links (Sommerville & Sawyer, 1997)	Techniques can be used for presenting and managing requirements as separate entities and describing and maintaining links between them, for example, during allocation, implementation, or verification.
	IBIS derivatives (Pinheiro, 2000) Document -centred models Database guided models (Pinheiro, 2000) RADIX (Yu, 1994) QFD (West, 1991)	Methods present traces and provide information to capture design rationale, for example, by providing automated support for discussion and negotiation of design issues.
	Languages, for example, ALBERT (Dubois, Du Bois, & Petit, 1994) or RML (Requirements Modeling Language) (Greenspan, Mylopoulos, & Borgida, 1994)	Traceability can be supported by using languages or notations.
Change management	Olsen's ChM model (Olsen, 1993) V-like model (Harjani & Queille, 1992) Ince's ChM model (Ince, 1994)	Change management process models and approaches.

- Generate predefined and ad-hoc reports.
- Generate documents that comply with standard industrial templates.
- Connect seamlessly with other tools and systems.

Conclusion

Requirements engineering for sociotechnical systems is a complex process that considers product demands from a vast number of viewpoints, roles, responsibilities, and objectives. In this chapter we have explained the activities of requirements engineering and their relations to the available methods. A large set of methods is available, each with their specific strengths and weaknesses. The methods' feasibility and applicability do, however, vary between phases or activities. Method descriptions also often lack the information of the methods' suitability to different environments and problem situations, thus making the selection of an applicable method or combination of methods to be used in a particular real-life situation complicated.

Requirements engineering deserves stronger attention from practice, as the possibilities of available methods are largely overlooked by industrial practice (Graaf, Lormans, & Toetenel, 2003). As requirements engineering is the process with the largest impact on the end product, it is recommended to invest more effort in industrial application as well as research to increase understanding and deployment of the concepts presented in this chapter.

This chapter has only listed a few examples of methods. For a more comprehensive listing of methods see Parviainen et al. (2003).

References

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002). *Agile software development methods: Review and analysis*. Espoo: Technical Research Centre of Finland, VTT Publications.

Ambler, S. W. (1998). *Process patterns. building large-scale systems using object technology.* Cambridge University Press.

Andriole, S. (1989). *Storyboard prototyping for systems design: A new approach to user requirements analysis.* Q E D Pub Co.

Ashworth, C., & Goodland, M. (1990). *SSADM: A practical approach.* McGraw-Hill.

Beck, K. (1999). *Extreme programming explained: Embrace change.* Reading, MA: Addison-Wesley.

Berlack, H. (1992). *Software configuration management.* John Wiley & Sons.

Björner, D., & Jones, C.B. (Eds.). (1978). *The Vienna development method: The meta-language: volume 61 of lecture notes in computer science*. Springer-Verlag.

Blanchard, B.S., & Fabrycky, W.J. (1981). *Systems engineering and analysis*. Prentice-Hall.

Booch, G., Jacobson, I., & Rumbaugh, J. (1998). *The unified modeling language user guide*. Addison-Wesley.

Bose, P. (1995). A model for decision maintenance in the winwin collaboration framework. *Proceedings of the 10th Conference on Knowledge-Based Software Engineering,* 105-113.

Bourdeau, R.H., & Cheng, B.H.C. (1995). A formal semantics for object model diagrams. IEEE Transactions on Software Engineering, 21(10), 799–821.

Chung, L., Cooper, K., & Huynh, D.T. (2001). COTS-aware requirements engineering Technique. *Proceedings of the 2001 Workshop on Embedded Software Technology* (WEST'01).

Davis, A. M. (1990). *Software requirements: Analysis and specification.* Prentice Hall.

Dobrica, L., & Niemelä, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering, 28*(7), 638-653.

Dorfman, M. (1990). System and software requirements engineering. In R.H. Thayer & M. Dorfman (Eds.) *IEEE system and software requirements engineering, IEEE software computer society press tutorial*. Los Alamos, CA: IEEE Software Society Press.

Dubois, E., Du Bois, P., & Petit, M. (1994). ALBERT: An agent-oriented language for building and eliciting requirements for real-time systems. Vol. IV: Information systems: Collaboration technology organizational systems and technology. *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences,* 713 -722.

Ericsson, K.A., & Simon, H. A. (1993). *Protocol analysis - revised edition*. MIT Press.

Gilb, T. (2003). *Competitive Engineering*. Addison-Wesley.

Girault, C., & Valk, R. (2002). *Petri nets for system engineering: A guide to modeling, verification and applications.* Springer-Verlag.

Gotel, O. (1995). *Contribution structures for requirements traceability*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London.

Graaf , B.S., Lormans, M., & Toetenel, W.J. (2003). Embedded software engineering: state of the practice [Special issue]. *IEEE Software magazine*, *20*(6), 61-69.

Greenspan, S., Mylopoulos, J., & Borgida, A. (1994). On formal requirements modelling languages: RML revisited. *Proceedings of ICSE-16, 16th International Conference on Software Engineering*, 135-147.

Hadel, J.J., & Lakey, P.B. (1995). A customer-oriented approach for optimising reliability-allocation within a set of weapon-system requirements. *Proceedings of the Annual Symposium on Reliability and Maintainability,* 96-101.

Harjani, D.R., & Queille, J.P. (1992). A process model for the maintenance of large space systems software. *Proceedings of Conference on Software Maintenance, Los Alamitos: IEE Computer,* 127-136.

Hatley, D.J., & Pirbhai, I.A. (1987). *Strategies for real-time system specification*. Dorset House.

Hooks, I., & Farry, K. (2001). *Customer-centred products*. Amacom.

Hull, M.E.C., Jackson, K., & Dick, A.J.J. (2002). *Requirements Engineering*. Berlin: Springer-Verlag.

HUSAT Reasearch Institute. (1998). *User centred-requirements handbook* (Version 3.2). [Handbook]. Loughborough, Leicestestershire, UK: Maguire.

Ince, D. (1994). *Introduction to software quality assurance and its implementation*. McGraw-Hill.

Institute of Electrical and Electronics Engineering, Inc. (1990). IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12).

Institute of Electrical and Electronics Engineering, Inc. (1998). IEEE Guide for Developing System Requirements Specifications (IEEE Std 1233).

Institute of Electrical and Electronics Engineering, Inc. (1998). IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830).

International Organisation for Standardisation. (Ed.). (1998). *Information technology software process assessment part 2: A reference model for processes and process capability*. (SPICE: ISO/IEC TR 15504-2). Geneva, Switzerland: ISO.

Juristo, N., Moreno A.M., & Silva, A. (2002). Is the European industry moving toward solving requirements engineering problems? *IEEE Software, 19*(6), 70-77.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff method. *Proceedings of the fourth IEEE International Conference on Engineering of Complex Computer Systems,* 68-78.

Komi-Sirviö, S., & Tihinen M. (2003, July 1-3). Great challenges and opportunities of distributed software development - an industrial survey. *Proceedings of Fifteenth International Conference on Software Engineering and Knowledge Engineering, SEKE2003,* San Francisco.

Kontio, J. (1995). OTSO: a systematic process for reusable software component selection, version 1.0. The Hughes Information Technology Corporation and the EOS Program.

Kotonya, G., & Sommerville, I. (1996). Requirements engineering with viewpoints. *Software Engineering Journal, 11*(1), 5-11.

Kotonya, G., & Sommerville, I. (1998). *Requirements engineering: process and techniques*. John Wiley & Sons.

Lang, M., & Duggan, J. (2001). A tool to support collaborative software requirements management. *Requirements Engineering, 6*(3), 161–172.

Lauesen, S. (2002). *Software requirements: styles and techniques*. Addison-Wesley.

Leffingwell, D., & Widrig, D. (2000). *Managing software requirements - a unified approach*. Addison-Wesley.

Matinlassi, M. & Niemelä, E. (2002). *Quality-driven architecture design method.* International Conference of Software and Systems Engineering and their Applications (ICSSEA 2002), Paris, France.

Mullery, G.P. (1979). CORE – A method for controlled requirement specification. *Proceedings of IEEE Fourth International Conference on Software Engineering.*

Nelsen, E. D. (1990). System engineering and requirement allocation. In R.H. Thayer & M. Dorfman, *IEEE system and software requirements engineering, IEEE software computer society press tutorial*. Los Alamos, CA: IEEE Software Society Press.

Olsen, N. (1993). The software rush hour. *IEEE Software Magazine*, 29-37.

Parviainen, P., Hulkko, H., Kääriäinen, J., Takalo, J., & Tihinen, M. (2003). *Requirements Engineering. Inventory of Technologies*. Espoo: VTT Publications.

Petri, C.A. (1962). *Kommunikation mit Automaten.* Bonn Institut für Instrumentelle Mathematik. Schriften des IIM Nr. 2.

Pinheiro, F. (2000). Formal and informal aspects of requirements tracing. *Proceedings of III Workshop on Requirements Engineering*, Rio de Janeiro, Brazil.

Pressman, R. S. (1992). *Software engineering, a practitioner's approach*, third edition. McGraw-Hill Inc.

Revelle, J.B., Moran, J.B., Cox, C.A., & Moran, J.M. (1998). *The QFD handbook*. John Wiley & Sons.

Robertson, S., & Robertson, J. (1999). *Mastering the requirements process*. Addison-Wesley.

Ropohl, G. (1999). Philosophy of sociotechnical systems. *Society for Philosophy and Technology 4*(3). Retrieved May 5, 2004, from *http://scholar.lib.vt.edu/ejournals/SPT/v4_n3pdf/ROPOHL.PDF.*

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE Wescon*, August 1970. Reprinted in Proceedings of 9th Int'l Conference Software Engineering 1987, 328-338, Los Alamitos: CA.

Sailor, J. D. (1990). System engineering: an introduction. In R.H. Thayer & M. Dorfman, *IEEE system and software requirements engineering, IEEE software computer society press tutorial*. IEEE Software Society Press.

Schneider, S. (2001). *The B-method: an introduction*. Palgrave.

Schoman, K., & Ross, D.T. (1977). Structured analysis for requirements definition. *IEEE Transactions on Software Engineering* 6–15.

Sheppard, D. (1995). *An introduction to formal specification with Z and VDM*. McGraw-Hill.

Shlaer, S., & Mellor, S. (1988). *Object-oriented system analysis: modeling the world in data* (Yourdon Press computing series). Prentice-Hall.

Siddiqi, J. (1996). Requirement engineering: the emerging wisdom. *IEEE Software, 13*(2), 15-19.

Sommerville, I., & Sawyer, P. (1997). *Requirements engineering: A good practise guide*. John Wiley & Sons.

Stevens, R., Brook, P., Jackson, K., & Arnold, S. (1998). *Systems engineering - Coping with complexity*. London: Prentice Hall.

Suchman, L. (1983). Office procedures as practical action. *ACM Transactions on Office Information Systems*, 320-328.

Sutcliffe, A. (1998). Scenario-based requirement analysis. *Requirements Engineering Journal, 3*(1), 48-65.

Thayer, R. H., & Royce, W. W. (1990). Software systems engineering. In R.H. Thayer & M. Dorfman, M. (Eds.), *IEEE system and software requirements engineering, IEEE software computer society press tutorial*. Los Alamos, CA: IEEE Software Society Press.

Weiss, D., & Lai, C. (1999). *Software product-line engineering: A family-based software development process.* Reading, MA: Addison-Wesley.

West, M. (1991, May 1-7). Quality function deployment in software development. *Proceedings of IEEE Colloquium on Tools and Techniques for Maintaining Traceability During Design*.

Yu, W. D. (1994). Verifying software requirements: a requirement tracing methodology and its software tool-RADIX. *IEEE Journal on Selected Areas in Communications, 12*(2), 234 -240.

Endnote

[1] This chapter describes the requirements engineering process based on work done in the MOOSE (Software engineering methodologies for embedded systems) project (ITEA 01002). The authors would like to thank the partners in the MOOSE project (*http://www.mooseproject.org/*), as well as the support from ITEA and the national public authorities.

PAPER VI

A survey of existing requirements engineering technologies and their coverage

In: International Journal of Software Engineering and Knowledge Engineering (IJSEKE) 17(6), pp. 1–24.

International Journal of Software Engineering and Knowledge Engineering

A SURVEY OF EXISTING REQUIREMENTS ENGINEERING TECHNOLOGIES AND THEIR COVERAGE

PÄIVI PARVIAINEN

VTT, Technical Research Centre of Finland
P.O. BOX 1100, 90571 Oulu, Finland
Paivi.Parviainen@vtt.fi

MAARIT TIHINEN

VTT, Technical Research Centre of Finland
P.O. BOX 1100, 90571 Oulu, Finland
Maarit.Tihinen@vtt.fi

Received (17 December 2005)
Revised (9 June 2006)
Accepted (1 November 2006)

Requirements engineering is a process in which a most diverse set of product demands from a most diverse set of stakeholders has to be considered. Thus, requirements engineering is generally thought of as the most critical and complex process within the development of embedded systems. Over the years, a lot of requirements engineering research has been carried out and reported, but still it seems clear that the industry is struggling with requirements engineering. Why is that, and what should be done to support the industry in tackling its problems? Develop a new method, tailor the existing ones, or better inform the industry of what is available that could help them in their problems? To find some answers, we carried out an inventory of the available requirements engineering technologies, while also looking into their support for requirements engineering. This paper describes the survey and reports our findings indicating that what is most urgently needed is information and evidence of the applicability of the available technologies in different situations, though further development of the technologies is also required.

Keywords: Requirements engineering; embedded systems; requirements engineering technologies; requirement management; requirements management technologies.

1. Introduction

Requirements engineering (RE) is generally accepted to be the most critical and complex process within the development of embedded systems, see, for example, [1], [2] and [3]. One reason for this is that in requirements engineering the most diverse set of product demands from the most diverse set of stakeholders has to be considered, making the requirements engineering process both multidisciplinary and complex. Thus, methods, techniques and tools are needed for supporting requirements engineering in order to ensure effective development and high quality of the products.

Several focused surveys and papers about requirements engineering methods, their evaluation, and the current state of industrial practices have been published. For example,

Neill and Laplante [4] have studied how widely a set of technologies was used for requirements elicitation and modelling in industry. Ardis et al. [5] have evaluated seven specification languages against predefined criteria through the various phases of the traditional waterfall model of software development. Haywood and Dart [6], in turn, have developed seven different classes for requirements representing technologies (goal hierarchy, state chart, hypertext, domain network, use case hierarchy, logic and conceptual state machine) and criteria for each class, against which methods can then be analyzed. Furthermore, White [7] has presented a comparative analysis of eight methods for modelling complex computer systems. However, we could not find any published research giving the "big picture" of the available methods; that is, describing how the available methods support and cover the requirements engineering activities as a whole. Such a picture is important from two viewpoints. First, knowing what solutions are already available makes it possible to direct future research and technology development work to the problems not yet solved. Second, as there are many different technologies available in the market, finding out what is available and what requirements engineering activities they support is a laborious task, often not justifiable for industry in practice. Thus, an inventory of the available requirements engineering technologies, including an overview of them and their support for different RE activities as well as references for further information, is likely to make finding the potentially existing methods easier. Accordingly, the selection of methods for a certain situation can be done based on a wider base of technologies, giving a better chance of selecting the best available technology.

The focus of this paper is on understanding how well the available requirements engineering technologies cover the different requirements engineering activities. The purpose of the survey is not to determine or point out the weaknesses or strengths of the various methods. Instead, the aim is to gain an overall view of the available requirements engineering technologies and to find the needs for tailoring or combining the available methods and techniques in order to provide effective and feasible support for the different activities of the RE process as a whole. Furthermore, this paper discusses issues to consider when selecting a technology or a combination of methods for real-life organisations and projects, mapping the available methods by the requirements engineering activities they support.

The survey is based on an extensive inventory of the available technologies as discussed in literature; the results of the study are published in detail in [8] and [9]. The inventory was made as part of a large research project, called MOOSE[a], in the year 2003. We not only considered specific RE technologies, but also more generic software engineering technologies as they also often include support for the RE activities. Over one hundred refereed papers were studied for gaining as comprehensive a picture as possible of available RE technologies and their support for different requirements

[a] MOOSE (Software Engineering Methodologies for Embedded Systems)

engineering activities. Due to the large number of available technologies (including different methods, techniques, models and languages), an experimental evaluation would have been too large a task for the whole set. Thus, the survey presented in this paper is based on published experiences and descriptions found. A subjective evaluation was made by four research scientists; each of them making their own evaluation for each technology against predefined criteria based on available literature. The criteria were developed during several workshops within the research and industrial partners of the MOOSE project. First, the research partners defined general criteria and those criteria were then analysed with industrial partners of the MOOSE project. Secondly, system and software requirements engineering process [8], [9] was defined by the researchers and at the same time both phase and activity specific criteria were defined. These were then analysed and refined in project's workshop meetings with industrial partners. After the subjective evaluation done by the researchers, the results were reviewed by the industrial and research partners of the MOOSE project. In addition, some more detailed experimental evaluations of some of the methods were carried out and the results have been included in this survey. Examples of those cases were

- a study of how scenarios and Goal-Oriented Requirements Engineering (GORE) can be combined to achieve a stable set of requirements [10],
- experiences from scenario-based architecting where software architecture was created to meet the goals (and related requirements and qualities) by using Quality driven Architecture Design and Analysis (QADA) [11],
- experiences and lessons learned of using UML-RT to develop embedded printer software [12],
- a study of software requirements implementation and management [13] and
- experiences of managing evolving requirements in an outsourcing context [14];

This paper is organized as follows: First, in Section 2, background information related to the requirements engineering process and activities within it are introduced as a basis for the coverage analysis. Second, in Section 3, a general survey of available methods and techniques based on literature is presented, and in Section 4, detailed study against the defined criteria is introduced. Section 5 discusses issues affecting method selection and further development needs of existing methods. Last, the main findings of the survey are discussed in the conclusion.

2. Requirements Engineering Process

Requirements engineering is divided into two main groups of activities, requirements development and requirements management. Requirements development includes activities related to discovering, analysing, documenting and validating requirements, whereas requirements management includes activities related to maintenance, namely identification, traceability and change management of requirements. Traditionally, RE has been performed in the beginning of software development lifecycle [15], but at present, RE is understood to be an iterative process, closely linked with other system development phases, such as design, implementation and testing. Fig. 1 describes a

system and software requirements engineering process based on [16], [17] and [18]. As stated by Zowghi [19], writing down a generic sequential plan of activities that could adequately describe what has to occur in RE is very difficult. Thus Fig. 1 aims at illustrating and clarifying the iterative and nondeterministic nature of the RE process, and is mainly aimed at showing the relations of the different RE activities. The process was used as a framework for which technology-support was searched and analyzed.

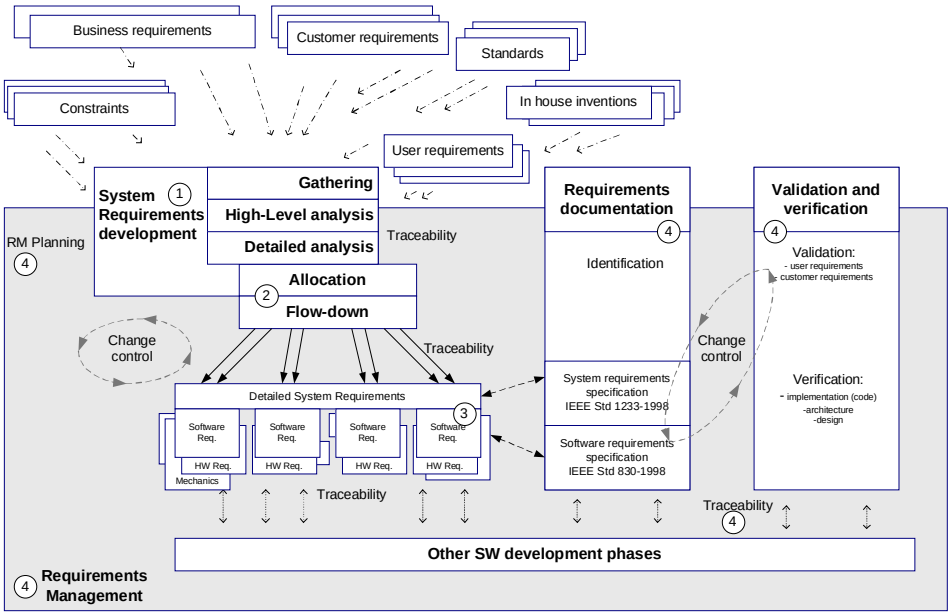


Fig. 1. System and Software requirements engineering [8]

Requirements engineering is an iterative process which will go into more detail during each iterative cycle. The main phases included in the RE process are (1) System requirements development, (2) Requirements allocation and flow-down, (3) Software requirements development, and (4) continuous activities.

The System requirements development (1) phase starts by gathering and eliciting requirements from various sources, e.g., users, customers, standards and marketing (see Fig. 1). The gathered raw requirements are first analysed at high level; the resulting high level analysis includes, e.g., a cost-benefit or technical feasibility analysis for a set of agreed raw requirements. After that, the requirements are analysed in detail, including trade-off analysis. In this step, the rationale behind the decisions is documented, and the system requirements are validated (continuous activities). In addition, traceability between requirement sources, raw requirements and system requirements needs to be established and maintained.

In the next phase, *allocation and flow-down* (2), the captured requirements will be allocated to system components and the detailed system requirements are defined and

validated. Allocation is architectural work carried out in order to design the structure of the system and to issue the top-level system requirements to subsystems. Flow-down consists of writing requirements for the lower level elements in response to the allocation. When a system requirement is allocated to a subsystem, the subsystem must have at least one requirement (usually more) that responds to the allocation. In the allocation and flow-down phase, requirements identification and traceability have to be taken into account and both system and software requirements specifications will be documented (continuous activities). Note that allocation and flow-down may be done for several hierarchy levels and that it starts as a multi-disciplinary activity, i.e., subsystems may contain hardware, software, mechanics but they are first considered as one subsystem. As the allocation and flow-down proceeds down the hierarchy levels, mono-disciplinary subsystems are defined.

During *software requirements development* (3) phase, the requirements set for the software are analyzed. Typically, both functional and quality aspects of the software system are modelled, and the contents of the subsystems are defined. Furthermore, the phase includes validation, verification and documentation of SW requirements specification, along with establishing and maintaining traceability, and change management (continuous activities).

Continuous activities (4) include requirements documentation, requirements validation and verification and requirements management. The analyzed requirements need to be documented to enable communication with stakeholders and future maintenance of the requirements and the system. Describing the relations of the requirements to each other is also a part of requirements documentation. Requirements validation consists of actions targeted to confirm that the behaviour of the developed system meets the user needs. Requirements verification consists of those actions aiming at confirming that the product of the system development process meets its technical specifications [20]. Requirements management is a continuous and cross-section process within requirements engineering. Requirements management begins along with planning and continues through the different activities related to identification, requirements traceability, and change control during and after the requirements development process. Requirements management activities, especially concerning traceability, should be performed as a part of the requirements development process, not as a separate task afterwards.

3. General Analysis of the Available RE Technologies

This section discusses the general analysis of the available RE technologies based on literature studies, i.e., published experiences and descriptions of the technologies. Because of the multi-disciplinary dimensions of RE, and since RE is a long-lasting process within SW development, the gathered RE technologies cover various methodologies, methods, techniques and languages. We did not only consider specific RE

technologies, but also more generic software engineering technologies as they also often include support for the RE activities. The purpose of the study was to provide as broad a view as possible on the different technologies available. The results of the analysis were evaluated by both industrial and research partners of the MOOSE project in workshops and reviews. Detailed descriptions and references of the analysed technologies are published in [8].

The criteria used include three categories: 1) general, 2) availability of information and 3) RE process coverage. The criteria are a result of several workshops with Moose project partners, aiming at defining criteria that could be used to analyze all RE technologies. This was not a straightforward task because the available RE methods, techniques or languages are very different and heterogeneous; they have been developed for different purposes and from different perspectives and even for different disciplines, which makes comparing them a complex exercise. Therefore, also sub-criteria were defined for each general criterion (see appendix A). The criteria used are introduced in Table 1.

Table 1. Criteria used in general analysis

General criteria:	
Established and well known	Is the technology widely known and actively studied and tested?
Understandable and usable	Does the technology have an understandable structure and provide guidelines of use in order to ensure effective utilisation also among inexperienced users?
Flexible and modifiable	Is the technology applicable to different environments and purposes of use?
Embedded viewpoint	Does the technology take the special characteristics of embedded software development into account?
Support for different requirement types	Is the technology suitable for handling different types of requirements from different sources?
Availability of up-to-date publications (~Publication year):	
Up-to-date information	Are there recent scientific publications related to the technology available?
Up-to-date experiences	Are there recently published experiences on the use of the technology available?
RE process coverage:	
Phase coverage	How well does the technology cover the different phases of

	the RE process?
Activity coverage	How well does the technology cover the different activities of the RE process phases?

In this paper, the term shortcoming is used to denote the inability of a requirements engineering technology, method, technique or approach to fulfil the set criteria. In a broader sense, the term is also used when referring to the overall limitations of the set of technologies to cover and provide support for the RE process. The shortcomings analysis is a subjective, and by no means a comprehensive one. The analysis presented in this paper is based on published experiences and descriptions of found journals and papers. The evaluation is of subjective nature and it was made by four research scientists; each of them making their own evaluation for each technology against the criteria described above. After the subjective evaluation, the results were also reviewed by the industrial and research partners of the MOOSE project. Due to the large number of available RE technologies, an experimental evaluation would be too large a task for the whole set. Experimental evaluation is, however, important and should be carried out for a few technologies. A number of experimental evaluations of some of the methods have been published (e.g., [10], [11], [12], [13] and [14]); the results of which have also been included in the analysis. The following table (Table 2) presents how each technology meets the set criteria.

Table 2. Technologies, their scope and availability of recent information

	GENERAL CRITERIA					RE PROCESS COVERAGE - Phases /Activities											Up-To-Date Publication	
Legend: F: Full, P: Partial, N: None, N/A: Not applicable, X: support, SW: support to software reqs						System Requirements Development					Allocation & Flowdown		SW requirements development					Publication year
METHOD	Established & Well-known	Understandable & Usable	Flexible & Modifiable	Embedded viewpoint	Support for different requirement types	Gathering	Top level analysis	Detailed analysis	Documentation	Validation&Verification	Allocation	Flow-down	High level analysis	Detailed analysis	Documentation	Validation&Verification	Change mgmt	Up-to-date information	Up-to-date experiences
ATAM	F	F	P	P	P			X			X	X						2002	2002
B-method	F	P	N	N					X	X					X	X		2001	1999
Booch	P										X		X	X		X		1995	1997
CARE	N				N													2002	--
CORE	P	P	P	P	F	X									X			1979	1992
Ethnography	P	F	P	N/A	P	X	X			X								1999	2001
Extreme progr. (agile)	F	F	F	N/A	P	X				X			X	X		X	X	2002	2002
Hatley-Pirbhai	F	F	P	P	F		X	X	X	X	X						X	2000	1994
HOORA	P	F	F	N	F	SW					X		X	X	X			2003	1997
Jacobson	N					SW					X		X	X	X	X		1992	--
JAD	P	P	P	N/A	N	X												1995	1998
MPARN	N			N/A	P		X											2002	--
OMT	P	F		N	P								X	X				1999	1995
OTSO	N	F	P		N													1996	1996
Petri Nets	F	P	P	P	P		X	X	X	X			X	X	X	X	X	2002	2002
Planguage	P	F	F	N/A	F		X	X	X				X	X	X		X	2003	--
Protocol an.	N					X												1993	--
QFD	F	F	F	N/A	P	X	X	X	X	X							X	2002	2002
REVEAL	N					X	X										X	2001	--
RUP	F	F		N/A		SW							X	X	X	X	X	2003	--
SADT	P	F	P	N	N									X	X	X		1993	2000
SASS	N	F	N	P	P			X						X				1979	--
SCR	N	F		P	P			X	X	X				X	X	X		2000	2000
SCRAM	N	P	N		N	X	X	X	X	X					X		X	1998	1998
Shlaer-Mellor	P	F	N	P	F								X	X	X	X		1998	1998
SRA	P	P		N	P		X				X	X						1995	--
SREM	N	P		N	P	SW							X	X	X	X		1990	1985
SSADM	P	P		N	P			X	X	X				X	X	X		2002	--
SSM	F	F	P	N/A	P	X												2001	1997
Storyboarding	P	F	P	N	P	SW							X			X		1992	1992
UML	F	F	P	P	P	SW							X	X	X			2002	2002
VDM	F	P							X	X					X	X		1997	2000
WinWin	F	F		N/A	P		X											2000	2001
Volere	F	F	F	N	P	X	X	X	X	X						X	X	2001	--
VORD	P	F	P	N	F	X	X	X	X	X					X			2000	1999
VOSE	N						X											1994	--
Z	F	P	N	N										X	X	X		2002	2000

Table 2 shows that the technology coverage is not sufficient for all the defined requirements engineering activities. For example, there are only two methods (ATAM, Architecture Tradeoff Analysis Method [21] and SRA, System Requirements Allocation Methodology [22]) providing support for performing requirements flow-down, and from which one (namely, SRA) showed no experiences of use, and the information found was outdated. On the other hand, there were plenty of supporting methods available for some phases of the RE process (e.g., gathering and documenting software requirements).

Furthermore, it was found that none of the methods were able to cover all the RE process activities; thus in order to achieve full support, combinations of technologies are needed. In practice, however, the technologies are of stand-alone type and independent, and very few descriptions of their relations or applicability with other methods or techniques was found in literature. Consequently, it proved difficult to gain an understanding of how to form feasible and effective combinations of the technologies.

The technology descriptions found in the literature were not precise, thus in order to be able to use the methods in practice, interpretation was needed. The overall comparison of the technologies also proved to be difficult, due to the fact that they had different focus areas and purposes of use, and they had been described at different levels of granularity in literature. For example, some technology descriptions only presented suggestive phases to be followed, while others described detailed steps, tasks and roles and provided different tools and templates to facilitate the support and performance of the tasks.

The results of the shortcomings analysis (Table 2) can also be used to assist organisations in the selection of suitable technologies by, for example, first identifying the needs of the company and then selecting applicable technologies for further analysis with the help of the table.

4. Detailed Analysis against Process Activities

In this section, the selected set of technologies is analyzed in more detail. The selected technologies are shown in the white rows in Table 2. The selection of technologies[b] for detailed analysis was mainly based on the availability of information; technologies showing more up-to-date information were favoured over those with older information, and technologies with published experiences on their use (especially from embedded systems) were favoured over those without any published experiences. Covering every phase of the RE process was also considered equally important. In addition to the basic rationale, also other aspects affected the selection; from among the different technology groups (e.g., formal methods) only few – those meeting the rationale best – were selected and some specific-purpose technologies with little information were left out, e.g., Soft System Methodology (SSM) [40] and WinWin [41].

[b] The technologies included in the analysis are ATAM [21], Ethnography [23], Extreme programming XP [24], Hatley-Pirbhai [25], HOORA Hierarchical Object Oriented Requirement Analysis [26], PetriNets [27], Planguage [28], Quality Function Deployment QFD [29], Rational United Process RUP [30], Structured Analysis and Design Technique SADT [31], Software Cost Reduction method SCR [32], Scenario-based Requirements Engineering SCRAM [33], Shlaer-Mellor Object-Oriented Analysis Method [34], Structured System Analysis and Design Methodology SSADM [35], Unified Modeling Language UML [36], Volere [37], Viewpoint-Oriented Requirements Definition VORD [38] and Z –language [39].

Next, each of the main RE process activities and the technologies supporting them are discussed separately. This is accompanied by a subjective evaluation of the level of support of the technologies for each activity. The analysis is based merely on literature, i.e. published information describing the technologies and experiences of their use. The tables summarise the support using the single X for limited support and the double X, (XX), for considerable support. A blank cell suggests that no corresponding evidence of support has been found in literature. The rating X is meant to indicate that some evidence of support for the criterion was found in the description of the method or in published experiences and the rating XX is meant to indicate that clear support, e.g., steps to carry out that support the criterion was found. The rating and rationale for the rating was first given by four research scientists; each of them making their own separate evaluation for each technology against predefined criteria. The agreed evaluation results were then also reviewed by the industrial and research partners of the Moose project. For example, for systems requirements engineering, Ethnography was given double X, (XX), in the column "Gathering /eliciting requirements from various sources". The rationale for this was that ethnography uses an ethnographer observing the users carrying out their normal work, and in this way eliciting user requirements (especially those related to work procedures and tasks to be performed). Ethnography also includes three presentation viewpoints and a presentation framework, which have been designed to facilitate communication between ethnographers and designers by structuring the ethnographic data. In Extreme Programming, by contrast, customer requirements are gathered using story cards, in which the customers write out the features they wish to have in the program release. This was rated with one X for the reason that only customer requirements are gathered. The ratings are not meant to be explicit, but they are merely given so as to support industrial companies in the selection of technologies for further analysis.

4.1. *System Requirements Development*

In this subsection, a detailed analysis of the selected technologies support for system requirements development phase is presented. The activities of this phase include:

- Gathering and eliciting requirements of different types from various sources, i.e., Constraints, Business requirements, Customer requirements, User requirements, Standards, In-house ideas.
- Describing the raw requirements in a way that is understandable to all stakeholders.
- Performing cost-benefit analysis for the requirements.
- Communication between stakeholders, i.e., requirement sources, developers and decision-makers.
- Prioritizing requirements.
- Recording a rationale of decisions made (also for those requirements that are left out).

- Documenting system requirements, including requirement dependencies, requirement identification and traceability to raw requirements, and technical description of requirements.
- Validating the requirements, for example, validating against requirement sources, ensuring un-ambiguity, or detecting possible conflicts, overlaps and dependencies.
- Making the final selection of requirements for the product.
- Managing requirement changes throughout the product lifecycle.

The studied technologies and the analysis results for the system requirements development phase are presented in Table 3. Technologies primarily designed for software development activities (e.g., XP and RUP) but also providing support for system requirements development activities are also included here.

Table 3. Support of selected technologies to system requirements development activities.

	Gathering /elicitation from various sources	Describing in understandable way	Cost-benefit analysis	Facilitates communication	Priorisation	Recording of rationale	System requirements documentation	Verification & Validation	Final selection of requirements	Managing requirements changes	Average support of technology (%)
Ethnography	XX	XX		XX				X			35 %
Extreme programming	X				X			XX	X	X	30 %
Hatley-Pirbhai		XX					XX	X		XX	35 %
Planguage			X	X		XX	XX			X	35 %
QFD	X	XX			XX	X	XX	XX	XX	XX	70 %
RUP		X		X			X	X		XX	30 %
SCR							XX	XX			20 %
SCRAM	X			X	X	XX	X	XX		X	45 %
Scrum				XX	X		X	X	X	X	35 %
SSADM			X		X			X	X		20 %
UML		XX					XX				20 %
Volere	XX	X		X	X	XX	XX		XX	XX	65 %
VORD	XX	XX			XX	XX	XX	X			55 %
Average support of criteria (%)	35 %	46 %	8 %	31 %	35 %	35 %	65 %	54 %	27 %	46 %	

Legend: XX = Provides considerable support to the activity
X = Provides some support to the activity

As Table 3 shows, there are several technologies that provide some level of support for system requirements development activities. The most supported activities are system requirements documentation, validation and management of changes. For validation of requirements (especially software requirements), also techniques such as consistency checking [42], default reasoning [43] and metrics [44] can be used. On the other hand, only few technologies cover cost-benefit analysis, recording a rationale and performing the final selection of requirements. In addition, only few technologies give support over half of the system requirements development activities: those are QFD (70%[c]), Volere (65%) and VORD (55%).

[c] Average support percentage (%) was calculated based on rating as follows: no support (empty cell) was given a value zero (0), some evidence of support (rating with X) was given a value one (1) and clear support (rating with XX) was given a value two (2). The value two (2) was the maximal value in each cell.

4.2. *Requirements Allocation and Flow-down*

In this subsection, a detailed shortcomings analysis is presented from the viewpoint of requirements allocation and flow-down.

Requirements Allocation. The activities that should be covered by requirements allocation methods cover include:

- Performing trade-off analysis.
- Management of non-allocable requirements, i.e., items such as environments, operational life and design standards, which apply unchanged across all the elements of the system or its segments.
- Allocation and management of non-functional (or quality) requirements (e.g., performance, reliability).
- Verification of requirement allocation, including verifying that each requirement is allocated to at least one subsystem and each subsystem has at least one requirement allocated to it.
- Management of changes to the initial system requirements.

Note: Allocation is architectural work, thus not all of its aspects are evaluated from the viewpoint of RE.

The requirements allocation technologies included in the shortcomings analysis are listed in Table 4. The table summarizes the technology support to the activities of the requirements allocation phase.

Table 4. Support of selected technologies to requirements allocation activities.

	Trade-off analysis	Management of non-allocable requirements	Management of non-functional requirements	Verification of requirement allocation	Management of changes	Average support of technology (%)
ATAM	XX		XX			40 %
Hatley-Pirbhai			XX	XX	XX	60 %
HOORA				XX		20 %
Average support of criteria (%)	33 %	0 %	67 %	67 %	33 %	

Legend: XX = Provides considerable support to the activity
X = Provides some support to the activity

As Table 4 shows, there are only three methods that partially cover the requirements allocation activities. None of the studied technologies covered all of the allocation activities. Furthermore, none of the methods provides any support for the management of non-allocable requirements. However, software architecture analysis methods, which are outside the scope of this analysis, do provide a more comprehensive support for the allocation activities. Further information on software architecture analysis methods can be found in, for instance, the survey by [45].

Requirements Flow-down. Activities which should be covered by requirements flow-down technologies cover include:

- Identification and management of derived requirements, i.e., requirements caused by the collection and organisation of requirements into a particular system configuration and solution.

- Analyzing allocated requirements from the sub-system viewpoint.
- Recording a rationale of decisions made (also for requirements that are left out).
- Documenting sub-system requirements, including requirements identification & traceability to top-level system requirements, and recording the dependencies between requirements.
- Verification of requirements against top-level system requirements, ensuring un-ambiguity, performing consistency checking, and coverage of allocated requirements.
- Management of requirement changes.

We could only find one method – the Architecture Tradeoff Analysis Method (ATAM) – to provide support for the requirements flow-down activities, or to be exact, one of the activities. While support was provided for "the recording of a rationale", none the other flow-down activities were supported. Software requirements development is a part of the flow-down activity, but as flow-down refers to multidisciplinary subsystems here, and not only software subsystems, also technologies are analysed separately for the flow-down. A software specific analysis is presented in the next section.

4.3. *Software Requirements Development*

This subsection discusses the shortcomings analysis for software requirements development technologies. Activities which should be covered by software requirements development technologies include:

- Gathering and eliciting software requirements from various sources.
- Analyzing the software requirements in detail so that sufficient input for design can be provided, including defining interfaces with other (sub)systems, checking inter-dependencies and conflicts, analyzing contradictory requirements, if found, and identifying design constraints.
- Modelling the software system. Models are developed to help understand the requirements, to reveal inconsistencies and incompletenesses and to add information to natural language descriptions of requirements in requirements documentation.
- Verifying the software requirements specification, including verification against system requirements, ensuring unambiguousness, performing consistency checking, checking the coverage of allocated requirements, and ensuring the completeness of the specification.
- Documenting the software requirements specification, including, e.g., requirements identification & traceability to higher level requirements, and recording the dependencies between requirements.
- Managing requirement changes.

The shortcomings analysis for technologies supporting software requirements development is summarized in Table 5. Note that the criteria of gathering and eliciting software requirements from various sources has been analysed in section 4.1. In addition, the activity of managing requirement changes will be introduced in the subsection 4.4.

Table 5. Support of selected technologies to software requirements development activities.

	Analysing the requirements for SW at detailed level	Modelling the software system	Validation & Verification	SW requirements specification documentation	Average support of technology (%)
Extreme Programming				X	13 %
HOORA	XX	XX	X	X	75 %
Petri Nets	X	XX	XX	X	75 %
RUP	XX	X	X	X	63 %
SADT	XX	XX	X	X	75 %
SCR	X	XX	X	X	63 %
Shlaer-Mellor	XX	XX	XX	X	88 %
SSADM	X	X	X	X	50 %
UML		XX		XX	50 %
Z	X	X	XX	XX	75 %
Average support of criteria (%)	60 %	75 %	55 %	60 %	

Legend: XX = Provides considerable support to the activity
X = Provides some support to the activity

There are several technologies providing support for the different activities involved in the software requirements development process. The most widely supported activities are modelling the software system, analysing the requirements for software in detail, and verifying the software requirements specification. One of the least supported activities is documenting the software specification. However, other support for documentation is available in the standards defining the contents and characteristics of requirements specification, e.g., IEEE Std 1233 [46] and IEEE Std 830 [47].

4.4. *Requirements Management*

This subsection introduces a detailed analysis from the viewpoint of requirements management (RM). Within the MOOSE project, several RM related technologies were researched; the respective descriptions are available in [8]. A detailed analysis of the selected RM methods, techniques and models started with the definition of the evaluation criteria. The criteria were divided into three categories according to the main RM activities: 1) identification, 2) traceability and 3) change management. The used criteria included:

Identification

- Requirements and their versions should be uniquely identified (ID_1).
- Recording additional information about requirements should be possible (ID_2).
- Managing requirements hierarchically should be possible (ID_3).
- Baselining requirements should be possible (ID_4).

Traceability

- Allocating and tracing requirements in requirement levels should be possible (RT_1).
- Forward and backward traceability must be available (RT_2).
- Post- and pre-traceability must be available (RT_3).

- Traceability to additional information must be available (RT_4).

Change management

- Support for SW engineering life cycle (e.g., development or maintenance) (CHM_1).
- Basic phases of change management must be supported (CHM_2).

The following table (Table 6) summarizes the analysis of requirements management technologies.

Table 6. Technologies support to requirements management activities

	ACTIVITY										Average support
METHOD	ID_1	ID_2	ID_3	ID_4	RT_1	RT_2	RT_3	RT_4	CHM_1	CHM_2	of technology (%)
ID											ID
Significant numbering	X	X	X								15 %
Non-significant numbering	X										5 %
Structure based naming/numbering	X		X		X					X	20 %
Symbolic identification	X	X									10 %
Dynamic renumbering in document based requirements' identification	X		X		X	X	X				25 %
Database record identification (see also "automated traceability links)	XX	XX	XX	XX	XX	XX	XX	XX		X	85 %
Baselining requirements	X			XX							15 %
RT											RT
Cross reference						XX	XX	X		X	30 %
RT matrixes: Table					XX	XX	XX	X		X	40 %
RT matrixes: List					XX	X	XX			X	30 %
Automated traceability links (see also "database record identification")					XX	XX	XX	XX		XX	50 %
REMAP					X		X	X		X	20 %
Low- and High-end RT REFERENCE MODELS					X		XX	X		X	25 %
Contribution structures for requirements traceability							X	X		X	15 %
HYDRA							X	XX		X	20 %
Multiview++	XX	XX	X				XX	XX		X	50 %
VORD	X	XX	X			X	X	X		X	40 %
Hatley-Pirbhai					X		X			X	15 %
SCRAM							X				5 %
DSDM				X						X	10 %
RUP	X	X		X	XX		XX			X	40 %
XP	X	XX				X	X	X		X	35 %
QFD					XX	XX	XX	X		X	40 %
RADIX (RT methodology and its SW tool)	XX	XX					XX			X	35 %
RChM											RChM
Olsen's ChM model									X		5 %
V-like model									X		5 %
Ince's change management model									X		5 %
AMES process model									X		5 %
Spiral-like change management model									XX		10 %
Generic ChM process model									XX		10 %
Using CM for RM	XX	XX	XX	XX			X	XX		XX	65 %
Using metrics to support requirement change management										X	5 %

Legend: XX = Provides considerable support to the activity
X = Provides some support to the activity

Requirements identification is an essential activity for requirements management; it can be considered a pre-requisite for requirements traceability and change management. The basic systems for identification are significant and non-significant numbering. These systems and baselining are well-known concepts. However, it seems that identification has been given just minor attention in literature. This may be due to the fact that identification can be embedded into a certain technology or that it is quite a self-evident part of the documentation, and at the same time, it is an important pre-requisite for requirements traceability and change management.

Our survey shows that the basic techniques are well-known and provide good support for traceability. For example, pre- and post-traceability are supported when using traceability matrices or cross-referencing. These techniques are also applied in many practical solutions (e.g., tools) or approaches. For example, the RADIX [48] method uses the cross-reference traceability technique and QFD [29] utilizes traceability matrices.

Manual solutions for traceability, such as manual traceability tables and manual cross-referencing in requirements document, are rather error prone and hard to maintain. The automated traceability links technique enables easier traceability and reporting of information. In practice, this means that requirements are managed in, for example, the RM tool and the tool implements an appropriate traceability model (e.g., tailored REMAP [49] or RT reference model [50]). However, automation may require considerable system investment, development and maintenance effort.

The basic techniques do not introduce any information elements (items and relations) which need to be traceable, while they still provide means for tracing them. On the other hand, practical models aim to show which items and relations should be traced. According to the survey, they have just one or few implementations or show few case experiences and thus the evaluation of their applicability is difficult. Some models also make an attempt to dictate items which should be managed (e.g. MULTIVIEW++ [51]). Among all the models considered in this survey, the so-called "RT reference models" seem to have the best empirical and practical level of traceability models. The survey also shows that there are well-known methods that also include some features for requirements traceability (e.g., XP, QFD, and RUP).

Efficient requirements change management (ChM) presupposes requirements identification and traceability. Different ChM models and approaches have different focuses on the product life cycle. They provide relatively coarse models for ChM (usually just main steps) and need to be adapted for different organisations. It is, however, difficult to find information concerning the practical implementation of the models with respect of requirements management.

5. Discussion

As can be seen in the previous sections, there are plenty of requirements engineering methods and techniques available. The purpose of our analysis was not to compare the different RE / RM methods or techniques with each other, but to find out how well the different requirements engineering activities are covered by the existing methods, and to gain a picture of the available method support in general. This was done for two reasons; to help direct our future research work and to support industry by means of providing information of what is available. It was found that not all the activities of our reference requirements engineering process were covered. In general, the analysis revealed the following findings regarding the coverage of requirements engineering activities:

- Not all the different activities of the RE process have sufficient method support; this was found inadequate especially in the cases of requirements allocation and flow-down activities.
- None of the found methods cover all the phases of the RE process (system requirements development, allocation & flow-down and software requirements development), or all the activities within a phase. Thus, in order to achieve full support of the activities, the methods need to be combined.

- The methods are stand-alone and independent, and very few descriptions of their relations and applicability to being used with other methods were found in literature.

On the other hand, the method descriptions available in literature and the Internet were ambiguous:

- Some of the methods are obsolete with no new information or published experiences available.
- An overall comparison of the methods is difficult, because they have different focus areas and purposes of use, and they have been described at different levels of granularity in literature.
- The method descriptions found in the literature are not explicit, thus in order to be able to use the methods in practice, interpretation and further guidelines are needed.

Another issue, perhaps the greatest obstacle for the application of the new methods by the industry, is the availability of information and evidence of the applicability of the methods to different kinds of products, projects and environments. Unfortunately, most of the method descriptions were found to concentrate on defining the process, steps and deliverables of the method rather than discussing the applicability, scope or limitations of the method, or its practical adoption. According to Fowler and Swatman [52], another common weakness of the existing RE approaches is that their developers have been concentrating solely on the theoretical foundations, or pragmatics, throughout the entire development process. This shortcoming makes the selection of an appropriate method difficult.

In the following, we discuss issues affecting the selection of a method or a combination of methods to be utilised in real-life organisations and projects. Identifying the specific characteristics of a given environment, project or product (situation) facilitates the selection of a RE method conforming to the particular situation best. Next, several issues, gathered from literature and the workshops held during the project, are listed that can be used to characterise embedded system or software development projects and products. The list is divided into two main categories: 1) characteristics of the development project, and 2) characteristics of the product being developed. Examples on how the different characteristics affect the RE method are also presented.

Characteristics of the development project

- Total number of requirements. For example, prioritization of requirements becomes more important if the number of requirements is high. Likewise, the analysis of requirement relations and documenting and maintaining descriptions of the relations will become more complex if more requirements need to be addressed.
- Number of developers and/or stakeholders involved in the project. For example, documentation needs are higher if the number of developers or stakeholders is high. Change management procedures also need to be more formal if more developers and/or stakeholders are involved. Prioritization and agreement of requirements will

become more complicated if more stakeholders are involved, as there is a higher chance of conflicting interests.
- Policies and maturity of the existing requirements engineering process of the organisation. For example, the feasibility and investment cost of adopting new processes or tools depend on the maturity of the organisation and the existing process where the new process or tool needs to be integrated.
- Type of working environment (distributed, centralized). For example, documentation needs are higher if the work is distributed. Also, with larger teams and especially when operating in different sites, formal traceability policies are needed.

Characteristics of the product being developed
- Safety criticality of the product. For example, more formal practices are commonly accepted to lead to more reliable products. Critical systems also require better traceability policies to enable analyzing the impact of changes in detail.
- Real-time requirements and constraints. For example, requirements analysis methods should provide means for modelling state behaviour, timing and response time requirements, so that the real-time requirements can be taken into account as early as in the phase of requirements analysis.
- Length of the product lifecycle. For example, traceability policies should be more comprehensive when the system lifecycle is long.
- Amount of legacy data (both software and hardware). For example, requirements analysis should provide mechanisms for combining new design elements to legacy data easily or engineering legacy data to meet the new requirements.
- Software/hardware ratio. For example, if both SW and HW are involved, documentation needs to be understandable to both SW and HW developers, and change management should include procedures for changes between hardware and software requirements. In this case, hardware constraints on software (e.g., limited use of memory and power consumption) should also be taken into account in requirements development.

Future research should, in addition to providing new, more effective solutions to requirements engineering, provide information and evidence on the applicability of the methods for different situations.

6. Conclusions

A large number of methods, techniques and tools are already available for requirements engineering, as seen in the sections above. However, as the Moose industrial inventory has shown, in practice industry uses rather common software engineering methods and tools [53]. The reasons for this are, for example, that new technologies often fail to take legacy artefacts into account, new technologies are often not mature enough to be applied in real-life applications, there is no adequate evidence of the benefits of the new technologies, and the complexity of some new development

technologies prevents them from being applied in industrial environments [53]. Furthermore, some reported surveys have established that there is a gap between research and practice and that companies need guidelines on how to use the existing research results [1], [54], [55]. Another reason for not adopting available methods or tools is that industry does not necessarily know what is available and where the available methods are applicable, and thus, selecting and introducing a new technology in real-life projects appears too laborious and risky. The reasons are to be found in the wide variety of characteristics of the product and the development project, and also in the broad range of activities involved in the RE process, including the support needs for each of these activities. The results of the survey described in this paper can be used to support the selection of applicable technologies for industrial needs; this paper gives a starting point for finding the most suitable solution from the viewpoint of an individual organisation.

This paper has described a survey of available technology support for requirements engineering activities. Main findings of the survey indicate, that not all the different activities of the RE process have sufficient method support, none of the found methods cover all the phases of the RE process or all the activities within a phase, and the methods are stand-alone and independent, and very few descriptions of their relations and applicability to being used with other methods was found in literature. The aim of the work described in this paper was to provide a "big picture" of available technology support for RE, not to determine or point out the weaknesses or strengths of the various technologies. This "big picture" helps to direct future research efforts to the most critical areas, while also providing the industry with an insight into what solutions are available for their requirements engineering problems. Based on the survey, we conclude, that what is most urgently needed is information and evidence of the applicability of the available technologies in different situations, though further development of the technologies is also required.

The fact is, as Kaindl et al. [56] argue, that the RE practice is most likely to remain immature until, for example, the researchers build on what others have done, rather than invent yet another modelling technique, and more practical formalisations are available and adopted by practitioners.

Acknowledgements

This paper has been written within the MOOSE (Software engineering methodologies for embedded systems) project, which is an ITEA project number 01002. The authors would like to thank all MOOSE partners for their valuable comments regarding our research results during the project [57]. We would like to give special thanks to Hanna Hulkko, Jukka Kääriäinen and Juha Takalo at VTT [58] for their contribution on the research work of which this paper is a summary. Furthermore, we would also like to express our gratitude for the support of the ITEA project [50] and Tekes [60].

Appendix A. General criteria description

Criteria	Description
Technology is established and well known.	The technology is widely known and actively studied and tested to assure optimal results of use. - Is developed by a recognized source. - Previous experiences of technology's usage have been published (incl. usage environment descriptions). - Is well documented. - Complies with standards (both industrial and organizational).
Method is understandable and usable.	The technology has to have understandable structure and usage guidelines in order to assure effective utilization and avoid misinterpretations also among inexperienced users. - Has defined, structured and systematic steps to be taken. - Has a set of defined roles together with their areas of responsibility. - Defines and/or uses uniform terminology common to all stakeholders. - Support and training for adoption are available. - Has reasonable adoption effort. - Is tool supported? - Usage experience, adoption guidelines and usage guidelines are available.
Technology is flexible and modifiable	The technology has to be applicable to different environments and purposes of use. - Is modular, i.e., composed of interdependent parts, from which the applicable ones for the particular project can be selected? - Can be used together with other RE methods, tools and techniques. - Can be used in different application domains. - Possible adoption limitations, problems and technology's suitability for different application domains have been documented. - Adaptation effort and time in particular project can be predicted.

Technology takes the embedded viewpoint into account.	The technology takes the special characteristics of embedded software development into consideration:
	- Provides effective means for modeling:
	  - State behavior
	  - Timing and response time requirements
	  - Hardware/software interface

	- Interruption and exception handling - External behavior of software modules - Supports the engineering of real-time requirements (e.g., prioritization over other requirements) - Supports effective change management between SW and HW requirements - Provides high effectiveness to pre-release change identification. - Takes hardware constraints on software (e.g., limited memory usage and power consumption) into account.
Technology provides support for different requirement types.	The technology is suitable for handling different types of requirements from different sources. - Functional requirements and non-functional requirements. - Constraints (e.g. domain-specific, legacy SW & HW).
Technology has up-to-date information available.	Technology has up-to-date information available, e.g., - scientific publications related to the technology and - published experiences on the use of the technology,
Technology gives support for whole RE process	Technology gives support for - different phases of RE process and - different activities of RE process phases.

References

1. Juristo, N., Moreno, A.M., and Silva, A.A. 2002. Is the European Industry Moving Toward Solving Requirements Engineering Problems? IEEE Software 19(6): 70-77.
2. Komi-Sirviö, S, and Tihinen, M. 2003. Great Challenges and Opportunities of Distributed Software Development - An Industrial Survey. In proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering, SEKE2003, San Francisco, USA pp. 489 – 496.
3. Siddigi, J. 1996. Requirement Engineering: The Emerging Wisdom. IEEE Software 13(2): 15-19.
4. Neill, C.J., and Laplante, P.A. 2003. Requirements Engineering: The State of the Practice, IEEE Software 20(6): 40-45.
5. Ardis, M.A., Chaves, J.A., Jagadeesan, L.J., Mataga, P., Puchol, C., Staskauskas, M.G., and von Olnhausen, J. 1996. A Framework for Evaluating Specification Methods for Reactive Systems Experience Report, IEEE Transactions on Software Engineering 22(6): 378-389.
6. Haywood, E., and Dart, P. 1996. Analysis of Software System Requirements Models. In Proceedings of Australian Software Engineering Conference, IEEE Computer Society Press, pp. 131-138.
7. White, S.M. 1994. Comparative Analysis of Embedded Computer System Requirements Models. In Proceedings of the First International Conference on Requirements Engineering, IEEE Computer Society Press, pp. 126-134.
8. Parviainen, P., Hulkko, H., Kääriäinen, J., Takalo, J. and Tihinen, M. 2003. Requirements engineering, Inventory of technologies. VTT Publications 508, VTT Technical Research

Centre of Finland. Available from: http://www.vtt.fi/inf/pdf/publications/2003/P508.pdf (Available 09.06.2006).
9. Parviainen, P., Tihinen, M., van Solingen, R., and Lormans, M. 2005. Requirements Engineering: Dealing with the Complexity of Sociotechnical Systems Development, Chapter 1, pages 1-20 in J.L. Mate, A. Silva (eds): Requirements Engineering for Sociotechnical Systems, Idea Group Inc. in 2005
10. Delnooz, C., and Vrijnsen, L. 2003. Experiences with scenarios and Goal-Oriented RE. In proceedings of LACS2003 (Landelijk Architectuur Congres), Netherlands.
11. Delnooz, C., Vrijnsen, L., Somers, L., and Hammer, D. 2003. Experiences with Scenario-based Architecting. In proceedings of International Conference on "Software and System Engineering and their Applications", ICSSEA2003, Paris, France.
12. Dohmen, L. A. J. and Somers, L. J. 2002. Experiences and Lessons Learned Using UML-RT to Develop Embedded Printer Software. In Oivo, M. & Komi-Sirviö, S. (eds.) proceedings of PROFES'2002 (Product Focused Software Process Improvement), LNCS 2559, Springer-Verlag, pp. 475-484.
13. Jäälinoja, J., and Oivo, M. 2004. Software Requirements Implementation and management. In proceedings of the 17th International Conference on "Software and System Engineering and their Applications", ICSSEA'2004. Paris, France.
14. Lormans, M., van Dijk, H., van Deursen, A., Nöcker, E., and de Zeeuw, A. 2004. Managing evolving requirements in an outsoucring context: An industrial experience report. In Proceedings of IWPSE'04 (International Workshop on Principles of Software Evolution). pp. 149-158.
15. Royce, W.W. 1970. Managing the Development of Large Software Systems. Proceedings of IEEE Wescon. Reprinted in Proceedings 9th Int'l Conference Software Engineering (1987). IEEE Computer Society Press, Los Alamitos, California, pp. 328-338.
16. Kotonya, G., and Sommerville, I. 1998. Requirements Engineering: Process and Techniques. John Wiley & Sons.
17. Sailor, J.D. 1990. System Engineering: An Introduction. In IEEE System and Software Requirements Engineering. Edited by Thayer, R.H., and Dorfman, M. IEEE Software Computer Society Press, Los Alamitos, California, USA. pp. 35-47.
18. Thayer, R.H., and Royce, W.W. 1990. Software Systems Engineering. In IEEE System and Software Requirements Engineering. Edited by Thayer, R.H., and Dorfman, M. IEEE Software Computer Society Press. Los Alamitos, California, USA. pp. 77-116.
19. Zowghi, D. 2000. A Requirements Engineering Process Model Based on Defaults and Revisions. Database and Expert Systems Applications, Proceedings. 11th International Workshop. pp. 966-70.
20. Stevens, R., Brook, P., Jackson, K., and Arnold, S. 1998. Systems Engineering -Coping with Complexity. Prentice Hall, London.
21. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J. 1998. The Architecture Tradeoff Method. Proceedings of the fourth IEEE International Conference on Engineering of Complex Computer Systems, pp. 68-78.
22. Hadel, J.J., and Lakey, P.B. 1995. A customer-oriented approach for optimising reliability-allocation within a set of weapon-system requirements. Proceedings of the annual symposium on Reliability and Maintainability, pp. 96-101.
23. Viller, S., and Sommerville, I. 1999. Social analysis in the requirements engineering process: from ethnography to method. Requirements Engineering, 1999. Proceedings. IEEE International Symposium on, pp. 6 -13.
24. XP homepage. URL: http://www.extremeprogramming.org/ (Available 09.06.2006).
25. Hatley, D.J., and Pirbhai, I.A. 1988. Strategies for Real-Time System Specification. Dorset House, New York.

26. Gennaro, G. 1995. Hierarchical Object Oriented Requirements Analysis (HOORA). Preparing for the Future 5(4), European Space Agency. http://esapub.esrin.esa.it/pff/pffv5n4/genv5n4.htm (Available 09.06.2006).
27. Schneeweiss, W.G. 2001. Tutorial: Petri Nets as a Graphical Description Medium for Many Reliability Scenarios. IEEE Transactions on Reliability 50(2): 159 - 164.
28. Gilb, T. 2003. Competitive Engineering. Addison-Wesley.
29. Revelle, J.B., Moran, J.W., and Cox, C.A.1998.The QFD Handbook, John Wiley & Sons.
30. Kruchten, P. 1998. The Rational Unified Process, Addison-Wesley.
31. Schoman, K., and Ross, D.T. 1977. Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, pp. 6-15.
32. Kirby, J.Jr., Archer, M., and Heitmeyer, C. 1999. Applying Formal Methods to an Information Security Device: An Experience Report. Proceedings of the 4th IEEE International Conference on High Assurance Systems Engineering, pp. 81-88.
33. Sutcliffe, A. 1998. Scenario-Based Requirement Analysis. Requirements Engineering Journal 3(1): 48-65.
34. Shlaer, S., and Mellor, S. 1988. Object-Oriented System Analysis: Modeling the World in Data. Yourdon Press Computing Series, Prentice-Hall.
35. Ashworth, C., and Goodland, M. 1990. SSADM: A Practical Approach. McGraw-Hill.
36. Booch, G., Jacobson, I., and Rumbaugh, J. 1998. The Unified Modeling Language User Guide. Addison-Wesley.
37. Robertson, S., and Robertson, J. 1999. Mastering the Requirements Process, Addison-Wesley.
38. Kotonya, G., and Sommerville, I. 1996. Requirements Engineering with Viewpoints. Software Engineering Journal 11(1): 5-11.
39. Sheppard, D. 1995. An Introduction to Formal Specification with Z and VDM. McGraw-Hill Publications
40. Checkland, P. 1981. Systems Thinking, Systems Practice, John Wiley & Sons, London.
41. Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R. 1998. Using the WinWin Spiral Model: A Case Study. IEEE Computer 31(7): 33-44.
42. Heitmeyer, C.L., Jeffords, R.D., and Labaw, B.G. 1996. Automated Consistency Checking of Requirements Specifications, ACM Transactions on Software Engineering and Methodology 5(3): 231-261.
43. Zowghi, D., Gervasi, V., and McRae, A., 2001. Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements, Eighth Asia-Pacific Software Engineering Conference (APSEC 2001), pp. 133-140.
44. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., and Theofanos, M. 1993. Identifying and Measuring Quality in a Software Requirements Specification, Proceedings of the First International Software Metrics Symposium, pp.141-152.
45. Dobrica, L., and Niemelä, E. 2002. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering 28(7): 638-653.
46. IEEE Std 1233, 1998. IEEE Guide for Developing System Requirements Specifications. Institute of Electrical and Electronics Engineering Inc.
47. IEEE Std 830, 1998. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineering.
48. Yu, W.D. 1994. Verifying software requirements: a requirement tracing methodology and its software tool-RADIX, IEEE Journal on Selected Areas in Communications 12(2): 234-240.
49. Ramesh, B., and Dhar, V. 1992. Supporting systems development by capturing deliberations during requirements engineering, IEEE Transactions on Software Engineering 18(6): 498-510.
50. Ramesh, B., and Jarke, M. 2001. Toward reference models for requirements traceability, IEEE Transactions on Software Engineering 27(1): 58 –93.

51. Toranzo, M., and Castro, J. 1999. Multiview++ Environment: Requirements Traceability from the perspective of different stakeholders, WER99 - II IberoAmerican Workshop on Requirements Engineering, Buenos Aires.
52. Fowler, D.C., and Swatman, P.A. 1998. Building information systems development methods: synthesising from a basis in both theory and practice. Software Engineering Conference, Proceedings. Australian, pp. 110-117.
53. Graaf, B., Lormans, M., and Toetenel, H. 2003. Embedded Software Engineering: state of the practice. IEEE Software 20(6): 61-69.
54. Davies, P., Sensors, T., & INCOSE RWG, The State of the Union in Requirements Engineering, INCOSE-UK Autumn Assembly 2001, http://www.incose.org.uk/Downloads/AA01-4-1_RE_Overview.pdf (Available 09.06.2006).
55. Nikula, U., Sajaniemi, J. and Kälviäinen, H., A State-of-the-Practice Survey on Requirments Engineering in Small- and Medium-Sized Enterprises, Technical report, 2000. Telecom Business Research Center (TBRC) http://www.tbrc.fi/pubfilet/TBRC_500000146.pdf (Available 09.06.2006).
56. Kaindl, H., Brinkkemper, S., Bubenko, J.A Jr., Farbey, B., Greenspan, S.J., Heitmeyer, L: Sampaio do Prado Leite, J.C., Mead, N.R., Mypoulos, J., and Siddiqi, J. 2002. Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda. Springer-Verlag London Limited. Requirements Engineering volume 7: 113-123.
57. MOOSE homepage. MOOSE (Software engineering methodologies for embedded systems), URL: http://www.mooseproject.org (Available 09.06.2006).
58. VTT homepage. VTT Technical Research Centre of Finland, VTT Electronics, URL: http://www.vtt.fi/ (Available 09.06.2006).
59. ITEA homepage. ITEA, Information Technology for European Advancement, URL: http://www.itea-office.org/ (Available 09.06.2006).
60. Tekes homepage. Tekes, National Technology Agency of Finland, URL: http://www.tekes.fi/eng/ (Available 09.06.2006).

PAPER VII

Experiences on analysis of requirements quality

In: Proceedings of the Third International Conference on Software Engineering Advances (ICSEA) 2008. Sliema Malta, 26–31 October 2008. Pp. 367–372.

Experiences on Analysis of Requirements Quality

Petra Heck
Consultant and Researcher
Laboratory for Quality Software (NL)
info@laquso.com, petraheck@hotmail.com

Päivi Parviainen
Senior Research Scientist
VTT Technical Research Centre of Finland
Paivi.Parviainen@vtt.fi

Abstract

The quality of any product depends on the quality of the basis of making it, i.e., the quality of the requirements has strong effect on the quality of the end products. In practice, however, the quality of requirement specifications is poor, in fact a primary reason why so many projects continue to fail. Thus, the current approaches as applied in practice are clearly not enough to develop high quality requirements specifications. Also, the poor quality of the requirements is typically not recognized during requirements development. In this paper we present a method called LSPCM developed for certifying software product quality. We also describe experiences from using the method for analyzing requirements quality in three cases. The three different cases show that the checks in the LSPCM are useful for finding inconsistencies in requirements specifications, regardless of the application domain.

1. Introduction

The importance and effect of software intensive systems increase continuously as more and more applications depend on the reliability, availability, and integrity of software. These systems are also becoming more and more complex, causing that the development of quality systems has become a major scientific and engineering challenge. The quality of the systems is strongly affected by the quality of the requirements but proper requirements engineering is not an established practice within the software developing community [1]. This causes that most errors are introduced in the requirements phase and are caused by poorly written, ambiguous, unclear or missed requirements, as reported by several studies [1, 2, 3]. Failure to correctly specify the requirements can lead to major delays, cost overruns, commercial consequences including the loss of money or property, layoffs, and even the loss of lives.

Requirements engineering (RE) is generally accepted to be the most critical and complex process within the development of embedded systems, see, for example, [4, 5, 6]. One reason for this is that in requirements engineering the most diverse set of product demands from the most diverse set of stakeholders has to be considered, making the requirements engineering process both multidisciplinary and complex.

In this paper we focus on verification of the requirements, meaning checking the quality of the requirements descriptions. We describe a method developed for certifying software product quality, focusing on the requirements part of the method. We also describe experiences from using the method in three cases in different application domains.

1.1. Requirements Quality

According to the IEEE Guide for Developing System Requirements Specifications [7], a well-formed requirement is "a statement of system functionality (a capability) that can be validated, that must be possessed by a system to solve a customer problem or to achieve a customer objective, and that is qualified by measurable conditions and bounded by constraints". Capabilities are the fundamental requirements of the system, representing the features or functions needed by the stakeholders. Conditions and constraints are attributes that are stipulated by a capability.

When requirements are expressed in natural language, descriptions should be written by using simple and concise language in order for them to be understandable for all stakeholders. However, in practice the quality of requirement specifications is poor, the requirements are ambiguous, incomplete, unverifiable, inadequately prioritized, and mutually inconsistent [8]. In fact, this poor quality of the requirements (incomplete and changing requirements) is a primary reason why so many projects continue to fail [9]. Thus, the current approaches as applied in practice are clearly not enough to develop high quality requirements specifications. Furthermore, the poor quality of the requirements is typically not recognized during requirements development; the requirements may be reviewed, but many of the defects are not found, causing that the defects replicate in the following work. The later these errors are found, the

978-0-7695-3372-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICSEA.2008.32

more impact they have on work products quality and the more costly they are to fix, i.e., according to Boehm [10] finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirement and design phase.

1.2. Related Work

Many checklists and questionnaires for ensuring the quality of requirements of varying degrees of completeness and usefulness are described in requirements engineering books [1, 11, 12, 13]. There are also several guidelines and standards published addressing the requirements quality [7, 14, 15, 16]. Thus, the theory of writing high quality requirements is well established. However, in practice, the requirements are developed by people who have had little or no training in requirements engineering [8], often lacking knowledge of the published theory. Also, evaluating requirements quality based on standards is not straightforward, as understanding the practical meaning of the terms used in the standards is not simple. There are also some tools that can help in the analysis of the quality of requirements written in natural language such as ARM [17], CICO [18] and QuARS [19]. These tools analysis is based on predefined quality criteria and elements of (English) language.

The LaQuSo Software Product Certification Model (LSPCM) was developed because we did not find any other practical method or framework for checking the quality of software artifacts, e.g., requirements specifications. We needed a method that would yield similar results regardless of who performed the quality check and which is also application domain independent. The framework is based on literature and own experiences, main new contribution being that the framework brings together information that was scattered over different sources.

2. LSPCM Overview

For the certification of software product quality the Laboratory for Quality Software (LaQuSo) has developed the Software Product Certification Model (LSPCM) [20]. Benefit from this certification is for example that the issues found during certification directly point to issues in the software product. Solving issues leads to a higher quality product. Another benefit of the certification is that in the certification the software product is investigated on a number of predefined criteria. These same criteria can later be used during the development of new products. In this way the organization learns from the certification: what is a good software product and how can we check our software products ourselves? The LSPCM has a similar structure as the well known models GQM (Goal Question Metric) and FCM (Feature Criteria Metric), as requirements are certified according to certification criteria (= Goal in GQM or = Feature in FCM), categories (= Question in GQM or = Criteria in FCM) and checks (= Metric in both GQM and FCM).

Software artifacts include in addition to the final software product (the source code or working system), also intermediate deliverables like user requirements and detailed design. Each major deliverable is a **Product Area** in the LSPCM. Each area can be further divided into subparts, called elements. These elements can be separate artifacts, a chapter within a document, or different parts of a larger model. For instance, the user manual will be a separate artifact delivered with the system, the non-functional requirements will be a separate section of the user requirements document, and the stakeholders can be described as part of the business process description.

Certification Criteria (CC) are criteria that apply to each Product Area (PA). For each of the CC's different Achievement Levels (AL) can be established. There are three CC, generic for all Product Areas:

- *Completeness (CC1):* All required elements in the Product Area (see section 2.2 for examples) should be present and as formalized as possible. The completeness of a PA can be basic or extra elements may have been added. These elements can be semi-formal (AL2) or formal (AL3), which refers to the fact that they are specified in a formal language. The more formal an element is, the easier it can be subject to formal verification (less transformation is needed).
- *Uniformity (CC2):* The style of the elements in the PA should be standardized. The uniformity of a PA can be only within the PA itself (AL1), with respect to a company standard (AL2), or with respect to an industry standard (AL3), meaning a general accepted description technique that is not specific for the company like the use of UML diagrams for design documents.
- *Conformance (CC3):* All elements should conform to the property that is subject of the certification. The conformance of the PA to a property can be established with different means that increase confidence: manually (AL1), with tool support (AL2), or by formal verification (AL3).

For each of these Certification Criteria different Achievement Levels can be established and the Certification Criteria can be translated into Specific Criteria (SC) per Product Area that indicate what completeness (CC1), uniformity (CC2), and conformance (CC3) mean for that Product Area. The Specific Criteria indicate what the required elements and checks are to achieve a certain level of the Certification Criteria. In this article we focus on the User Requirements Product Area of the LSPCM and the specific criteria for it.

2.1. Analysis Process

This section explains the general idea behind the LaQuSo Model, more details can be found in [20].

Although the current goal of the model is to hand out certificates to customers, its framework and checks can just as well be used by anyone that needs to check the quality of requirements. Next we will explain how LaQuSo hands out certificates.

If an organization wants confidence in a software artifact a LaQuSo certificate can be requested. Being part of universities, LaQuSo is an independent evaluator. Certification is a check that the artifact fulfils a defined set of properties and the certificate will always refer to the properties that were used. The certificate covers only the product quality, not the management and development processes. The quality certificate consists of a diagnosis report and verdict document. Certification may also be an iterative process where the customer delivers improved versions of an artifact until a certificate is achieved. Each certification case has its own goals and a specific certification plan (steps to take and techniques to apply) is made for each project. In this paper we describe three projects in which the goal was to grant a "Consistency of User Requirements" certificate. This certificate has the following characteristics:

Table 1. Certificate characteristics

Product Area	User Requirements
Properties	Consistency
Level	Manually verified
Description	Check on the internal consistency of the requirements
Input	Natural language requirements specification

Next, we'll describe the elements we consider to be part of a complete user requirements specification:

- *Functional requirements or Use-cases.* Functional requirements describe the functionality of the system from the perspective of the user in plain text or in the form of use-cases. A use-case is a named "piece of functionality" as seen from the perspective of an actor. For each use-case several use-case scenario's are given (sequences of actions, events or tasks), the permitted or desired ones as well as the forbidden ones.
- *Objects.* Many types of entities play a role in the environment processes and some of them have to be represented in the system. Only these are collected. The object description can be informal in the form of a glossary, or more advanced in the form of a data dictionary or object model.
- *Behavioral Properties.* General behavioral properties that should hold, such as properties expressing that certain conditions may never occur or that certain conditions should always hold. Usually these properties have a temporal aspect and therefore it is possible to express them in temporal logic, although a translation in natural language is essential for most stakeholders.
- *Non-functional requirements.* A set of different types of quality attributes that can be used to judge the operation of a system.

2.2. Requirement Checks

In this section we suggest a necessarily incomplete list for the User Requirements PA. The specific criteria (SC) are a direct translation of the three general certification criteria to the user requirements PA: describe the requirements as detailed and as formal as possible (CC1 -> SC1), comply with requirements engineering standards (CC2 -> SC2), and maintain correct and consistent relations between the elements in the requirements description (CC3 -> SC3a) and with the context analysis (CC3 -> SC3b). From this set of elements and the specific criteria we derive a number of specific checks (based on literature, standards and our own experience).

Table 2. The requirement checks

Category	Check
SC1.1 Required Elements	• Functional requirements describe the functionality of the system from the perspective of the user. • Non-functional requirements i.e., quality requirements are described (e.g., performance and security measures). • Glossary defines the entities that have to be represented in the system.
SC1.2 Semi-formal Elements	• Data dictionary or object model that contains data elements' definitions and representations including semantics for data elements. The semantic components focus on creating precise meaning of data elements. • Use cases (with scenarios) are defined and for each use case several use case scenarios are given (sequences of actions, events or tasks), the permitted or desired and the forbidden ones. • Behavioural properties, i.e., the constraints on the behaviour of the system are defined. The constraints express that e.g. certain conditions may never occur or certain conditions should always hold. Usually behavioural properties have a temporal aspect.
SC1.3 Formal Elements	• Relational diagram of data/object model is used to describe conceptual data models by providing graphical notations for entities and their relationships, and the constraints that bind them. Examples of diagramming techniques are UML class diagram and ER-diagram • Process model of use case scenarios describe the relations between the steps in the use case scenarios in a formal language like Petri Nets. • Behavioral properties specifications are expressed in a formal language. If they e.g. have a temporal aspect, temporal

	logic can be used.
SC2.1 Compliance w. Industry Standards	• ERD diagram for object/data model; • UML diagrams for use cases.
SC3a.1 Internal Correctness	• No two requirements or use cases contradict each other. • No requirement is ambiguous. • Functional requirements specify what, not how. • Each requirement is testable. • Each requirement is uniquely identified. • Each requirement is atomic. • The glossary definitions are non-cyclic. • Use case diagrams correspond to use case text.
SC3a.2 Automated Correctness Checks	• Requirements are stored in a requirements management tool which uniquely identifies them.
SC3a.3 Formally Verified Correctness	• Verify use case scenario models for e.g. correct workflow (no deadlocks or dead tasks) and mutual consistency. • Check data model diagram for normal form.
SC3b.1 External Consistency	• Each ambiguous or unclear term is contained in the glossary. • The use cases or functional requirements are a detailing of the environment description in the context analysis (no contradictions). Each step in a business process that involves the system has been included. Each task that the system should fulfil for its environment has been included. All actors of the context analysis have been included in the requirements. • Each object is mentioned in the requirements and all objects mentioned in the requirements are contained in the object model. • The requirements do not contradict the behavioural properties. • The use case or functional requirements do not conflict with the non-functional requirements.
SC3b.2 Automated Consistency Checks	• Requirements and glossary/objects are stored in a requirement mgmt tool showing the requirements, scenarios, actors, and objects relations.
SC3b.3 Formally Verified Consistency	• Verify use case scenario models for e.g. compliance with behavioral properties and non-functional requirements. • Verify that the requirements description complies with the environment description from the context analysis.

3. Case experiences

In this section we describe three cases where the method has been used to analyse requirements. For each of the cases, the findings are presented on a general level, including a reference to the check category that resulted in the finding.

3.1. Central Registration System

The system is a central point where new identification numbers are generated, distributed and registered. We were asked to judge the quality of the functional design, which consisted of functional requirements, 15 use case descriptions with UML activity diagrams, a process model of the business processes, a functional architecture (logical module structure), an object model, a glossary and a supplementary specification (all non-functional requirements such as legal, security, performance etc.).

The following types of inconsistencies were found:

- A number of spelling and structural errors were found. [SC3a.1]
- Some post-conditions of use cases were not consistent with the main scenario. [SC3a.1]
- Activity diagrams did not use the correct (UML) symbols: e.g. included states as activities. [SC2.1]
- The object model did not use ERD symbols correctly and did not contain much description for the attributes. [SC1.3 and SC2.1]
- The glossary contained only abbreviations. [SC1.1]
- The activity diagrams did not always match the use case text (especially not for the alternative flows). [SC3a.1]
- One of the actors was not used in a consistent manner (mix between human and nonhuman). [SC3a.1]
- One use case mentions two options in the summary and illustrates only one in the scenarios. [SC3a.1]
- Use cases described system features that were not mentioned in the other documents. [SC3b.1]
- There was an overview document that did not contain all use cases and their relations. [SC3b.1]
- Some components to support the use cases were missing in the functional architecture. [SC3b.1]
- Use cases for administration functions such as user management were missing. [SC3b.1]

All major inconsistencies were solved before the design was handed over to the developers of the system. This minimized the input needed from the designers during the development phase and reduces the risk for confusion and misinterpretation.

3.2. Counter Automation Solution

The company operates many offices with multiple counters were customers come for various transaction types. The system is a central registration system for all

counter transactions that take place. Client applications support the counter workers. We were asked to judge the quality of the functional design, which consisted of 200+ use case descriptions with flow charts, supplementary specifications (all requirements that were not specific to one single use case) and a glossary.

The following types of inconsistencies were found:

- There were around 200 use cases without a clear overview of the use cases and their relations. [SC1.2]
- Use case structures were not always applied correctly. E.g. the name of a use case should be noun + verb and a use case should have a clear trigger. [SC2.1]
- Paragraphs in use cases that summarize information from the rest of the documents did not always include all relevant items. [SC3b.1]
- Not all referenced documents were included in the baseline. [SC1.1]
- There were many open points ("to be decided") in the documents. [SC3a.1]
- The overview and reference documents in the supplementary specifications were not consistent with the separate use cases. [SC3b.1]
- Many times it was not clear if 'N/A' (not applicable) was correctly filled in. [SC3a.1]
- The content of a document or paragraph did not always match its purpose. [SC3a.1]
- Not all use case documents had the same layout. [SC3a.1]
- Flowcharts were not always consistent with the use case text. [SC3a.1]
- The glossary did not contain all terms and missed a clear list of translations English – Dutch. [SC1.1]
- Many system management functions were not specified. [SC3b.1]
- The only structure in the document set was sequentially numbered per release (1 till 5) [SC3a.1]

With so many inconsistencies the document set was not suitable for system maintenance. It was not possible or very time-consuming to assess the impact of future change requests. The document set was adjusted according to the findings as much as time and budget permitted. The rest of the findings were used as cautions in the use of the documents for implementation of change requests.

3.3. Embedded Systems Case

The target of the analysis was system requirements for a large embedded system. This was also the first time the method was used by some one not familiar with the method beforehand.

The provided material included over 200 requirements. As the amount of requirements was high, not all of them could be checked according to the check list. Instead a sub-set of requirements was selected for the analysis. Altogether the analysis covered about 50 requirements. Requirements were selected randomly individually and as all requirements belonging to a feature.

The LSPCM method had so far been used mainly for software-only information systems and certain documentation was expected (user requirements, software requirements, high-level design). In this case, the checks defined by the method needed to be tailored. In practice, a combination of the checks for user requirements and high level design was done, however, not all checks of high-level design were included (as they were design issues). Also, in addition to the aspects indicated by the method, two additional checks were added based on discussion with the company's representative, namely stakeholders and acceptance criteria.

The following types of inconsistencies were found:

- Requirements relations / references to each other were unclear. [SC3a.1]
- No non-functional requirements were defined. [SC1.1]
- Various different ways to describe the requirements, from one line descriptions to several pages of use case steps, sequence charts, etc. [SC2.1]
- Lot of unclear abbreviations were used. [SC1.1, SC3a.1, SC3b.1]
- Not defining "what" but also a lot of "how", i.e., lot of design level issues. [SC3a.1]
- Ambiguities in requirement definitions. [SC3a.1, SC3b.1]
- No stakeholders defined for the requirements. [SC3a.1]

4. Discussion and Lessons Learnt

The three different projects show that the checks in the LSPCM are useful for finding inconsistencies in requirements specifications, regardless of the application domain. The checks that mostly found issues were the manual level checks [SC1.1, SC2.1, SC3a.1 and SC3b.1], as there were little formal elements defined for the requirements in all three cases. That some projects need tailoring of the LSPCM is already foreseen in the model; one could argue that the LSPCM is more a general framework to define specific checks per project than a rigid model. Within this general framework one can predefine rigid certificates by combining different checks.

As expected the three projects all lead to new general checks to be added to the list that is already included in the LSPCM. The idea is that this list will continue to grow with new experiences, literature and standards appearing.

The projects also taught us that requirements validation remains expert work. It is hard to explain in written text how to do this or what to look for. The idea of the LSPCM is to give some guidance to evaluators, but more details and examples need to be included in

the future to make it usable for people not so experienced in requirements engineering. For example, the statement "each ambiguous term is included in the glossary" will lead to different scores by each assessor, but at least it gives them a hint to pay attention to the link between the requirements text and the glossary.

For some of the projects it was necessary to first make a translation of the input material to the elements as described in the model. Each requirements document in practice will have a different structure. This means the start of each project needs to consist of creating a picture of the structure of the input requirements specification: what are the elements in the structure and what are their interrelations?

Another remarkable finding is that the better the quality of the input requirements, the more inconsistencies we can find. With really poorly written requirements, it is best to start again at the beginning working with the original analyst to identify the different elements in the specification and reconstruct them in a new document.

5. Conclusions

We have presented a method called LSPCM and experiences of using the method for certifying software product quality focusing on the requirements part of it. We have described experiences from using the method in three cases and described the general findings relating to the requirement descriptions quality. The three projects show that the checks in the LSPCM are useful for finding inconsistencies in requirements specification, regardless of the application domain. However, it is clear that requirements verification and validation remains expert work. The idea of the LSPCM is to give some guidance to evaluators, but more details and examples need to be included in the future to make it usable by people not so experienced in requirements engineering. That will be the focus of our future work.

References

[1] Firesmith, D. G., "Quality Requirements Checklist", in Journal of Object Technology, vol. 4, no. 9 November - December 2005, pp. 31 - 38, http://www.jot.fm/issues/issue_2005_11/column4

[2] Martin, J, 1984, An Information Systems Manifesto, Prentice Hall

[3] Damian D. 2002. The study of requirements engineering in global software development: as challenging as important. In Proceedings of Global Software Development, Workshop #9, organized in the International Conference on Software Engineering (ICSE) 2002, Orlando,FL.

[4] Komi-Sirviö, S, & Tihinen, M., 2005, Lessons learned by participants of distributed software development, Knowledge and Process Management, vol. 12, no. 2, pp. 108-122

[5] Juristo, N., Moreno, A.M., and Silva, A.A. 2002. Is the European Industry Moving Toward Solving Requirements Engineering Problems? IEEE Software 19(6): 70-77.

[6] Siddigi, J. 1996. Requirement Engineering: The Emerging Wisdom. IEEE Software 13(2): 15-19.

[7] "IEEE Std 1233 1998 Edition, Guide for Developing System Requirements Specifications," The Institute of Electrical and Electronics Engineers, Inc. 1998.

[8] Donald G. Firesmith: "Specifying Good Requirements", in Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 77-87. http://www.jot.fm/issues/issue_2003_07/column7

[9] The Standish Group International, Inc. The CHAOS Report, published on www.standishgroup.com 1996, 1998, 2000, 2002, 2004 and 2006.

[10] Boehm, 2001 B. Boehm, Software defects reduction top 10 list, IEEE Computer 34 (2001) (1), pp. 135–137.

[11] R. R. Young, The Requirements Engineering Handbook. Norwood, MA: Artech House, 2004.

[12] K. E. Wiegers, Software Requirements, 2nd edition. Redmond, Washington: Microsoft Press, 2003.

[13] Ian Sommerville and Pete Sawyer: Requirements Engineering: A Good Practices Guide, John Wiley & Sons, 1997.

[14] IEEE Std 830 1998 Edition, IEEE Recommended Practice for Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, Inc. 1998.

[15] IEEE Std 1061-1998, IEEE Standard for a Software Quality Metrics Methodology [23]

[16] IEEE Std 1362-1998, IEEE Guide for Information Technology System Definition Concept of Operations (ConOps) Document [1]

[17] ARM, http://satc.gsfc.nasa.gov/tools/arm/, Wilson, W. M., Rosenberg, L.H. & Hyatt, L. E., Automated Quality Analysis Of Natural Language Requirement Specifications, NASA, SATC

[18] CICO, http://circe.di.unipi.it/Cico/ e.g., Gervasi, V., & Nuseibeh, B., Lightweight validation of natural language requirements, Software: Practice & Experience, 32(2):113-133, Feb. 2002.

[19] QuARS, http://quars.isti.cnr.it/, Lami, G., QuARS: A Tool for Analyzing Requirements, Technical Report, CMU/SEI-2005-TR-014

[20] Petra Heck, Marko van Eekelen. The LaQuSo Software Product Certification Model, CS-Report 08-03, Technical University Eindhoven, 2008.

Series title and number
VTT Science 6

Title	**Global software engineering** **Challenges and solutions framework**
Author(s)	Päivi Parviainen
Abstract	The increasingly complex and competitive market situation has resulted in Global Software Engineering (GSE) becoming more and more common practice. Companies need to use their existing , as well as global resources Thus, the ability to collaborate effectively has become a critical factor in today's software development. The main expected benefits from GSE are improvements in development time, being closer to the customers and having flexible access to better specialized and less costly resources. In practice, however, the productivity in distributed software development drops up to 50 per cent compared to single site software development. Main reasons behind this productivity drop are misunderstood or mismatched processes between teams, and poor visibility into and control of the development activities at all sites involved. The purpose of this thesis is to analyse in more detail why this is the case and what could be done to improve the situation in practice in the companies' daily work. In this thesis, the challenges in GSE are discussed based on their root causes and then summarised into the GSE framework. The root causes are time difference and distance, multiple partners, lack of communication, coordination breakdown, different backgrounds, and lack of teamness and trust. Then solutions for these challenges are discussed from people, process and technology viewpoints and summarised into the GSE framework. As a more detailed example of challenges to a subprocess, requirements engineering (RE) in GSE is presented. RE is discussed similarly as the GSE in general, first challenges are discussed and then solutions to the challenges are presented. The work reported in this thesis is based on extensive empirical work, carried out over several years. The empirical work was carried out in several phases: in the first phase, an industrial inventory was made, including industrial experience reported in the literature. Based on this, an initial framework for GSE was developed, consisting of the main challenges to be addressed in GSE projects. After this first phase, two sets of industrial cases were carried out, addressing a wide set of GSE challenges by trying out the GSE solutions identified in companies and validating the GSE framework. Altogether, 52 industrial cases relating to distributed development were carried out during the projects over the years 2004–2011. This thesis shows that although GSE is common, it is still challenging and companies should carefully weigh the benefits and costs of doing the work in distributed setting vs. doing it single site. This thesis is a step towards better, more productive and higher quality GSE, as it helps companies to be aware and address potential challenges early via the GSE framework. The work presented also helps companies to find validated solutions to address the challenges in their practice.
ISBN, ISSN	ISBN 978-951-38-7459-9 (soft back ed.) ISSN 2242-119X (soft back ed.) ISBN 978-951-38-7460-5 (URL: http://www.vtt.fi/publications/index.jsp) ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)
Date	April 2012
Language	English, Finnish abstract
Pages	106 p. + app. 150 p.
Publisher	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland, Tel. 020 722 111

Julkaisun sarja ja numero
VTT Science 6

Nimeke	**Globaali ohjelmistokehitys** **Kehikko haasteista ja ratkaisuista**
Tekijä(t)	Päivi Parviainen
Tiivistelmä	Jatkuva tuotteiden monimutkaistuminen ja kiihtyvä kilpailutilanne ovat johtaneet siihen, että globaali ohjelmistokehitys (GSE) on yhä yleisempää. Globaalin ohjelmistokehityksen potentiaalisia hyötyjä ovat lyhyemmät tuotekehitysajat, läheisyys asiakkaan kanssa sekä mahdollisuus käyttää erikoistuneita ja/tai edullisempia resursseja joustavasti. Käytännössä hajautetun ohjelmistokehityksen tuottavuus kuitenkin laskee jopa 50 prosenttia paikalliseen ohjelmistokehitykseen verrattuna. Tämä johtuu mm. väärinymmärretyistä tai yhteensopimattomista prosesseista tiimien välillä sekä eri paikkakunnilla tehtävän kehityksen hallitsemattomuudesta. Tutkimuksen tarkoitus on selvittää tarkemmin, miksi näin tapahtuu ja mitä voitaisiin tehdä käytännössä tilanteen parantamiseksi yritysten päivittäisessä toiminnassa. Tässä työssä esitetään globaalin ohjelmistokehityksen haasteita ja niiden ratkaisuja. Haasteiden aiheuttajat esitetään, ja sitten haasteet esitetään osana globaalin ohjelmistokehityksen kehikkoa. Haasteiden aiheuttajia ovat aikaero ja etäisyys, useat osapuolet, kommunikoinnin puute, hallinnan hajautuminen, erilaiset taustat ja tiimiyden ja luottamuksen menetys. Tutkimuksessa myös esitetään ratkaisuja näihin haasteisiin ihmisten, prosessin ja teknologian näkökulmasta, ja myös ne liitetään mukaan globaalin ohjelmistokehityksen kehikkoon. Esimerkkinä osaprosessin näkökulmasta esitetään vaatimusmäärittely ja -hallinta globaalissa ohjelmistokehityksessä. Tutkimus perustuu laajaan empiiriseen aineistoon, jota on koottu usean vuoden aikana. Empiirinen työ tehtiin useassa osassa. Ensimmäisessä vaiheessa tehtiin yritysten GSE-käytäntöjen nykytilan selvitys sisältäen kirjallisuudessa raportoidut yritysten kokemukset. Tämän perusteella laadittiin ensimmäinen versio globaalin ohjelmistokehityksen kehikosta, joka sisälsi päähaasteet, jotka tulee ottaa huomioon globaaleissa ohjelmistokehitysprojekteissa. Ensimmäisen vaiheen jälkeen vietiin läpi kaksi joukkoa teollisia tapaustutkimuksia. Nämä tutkimukset kohdistuivat laajaan joukkoon globaaliin ohjelmistokehitykseen liittyviä asioita. Tapaustutkimuksissa kokeiltiin ratkaisuja yrityksissä tunnistettuihin haasteisiin ja samalla validoitiin globaalin ohjelmistokehityksen kehikkoa. Yhteensä vietiin läpi 52 teollista tapaustutkimusta vuosien 2004–2011 aikana useassa eri projektissa. Tutkimus osoittaa, että vaikka GSE on yleistä, se on edelleen haastavaa ja yritysten täytyy huolellisesti punnita sen mahdollisia hyötyjä ja kustannuksia verrattuna paikalliseen kehittämiseen. Tämä työ on askel kohti parempaa, tuottavampaa ja laadukkaampaa globaalia ohjelmistokehitystä, sillä se auttaa yrityksiä huomaamaan mahdollisia ongelmia ja reagoimaan niihin aikaisin käyttämällä globaalin ohjelmistokehityksen kehikkoa. Tulokset myös auttavat yrityksiä löytämään hyviä ja kokeiltuja ratkaisuja käytännössä kohtaamiinsa ongelmiin.
ISBN, ISSN	ISBN 978-951-38-7459-9 (nid.) ISSN 2242-119X (nid.) ISBN 978-951-38-7460-5 (URL: http://www.vtt.fi/publications/index.jsp) ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)
Julkaisuaika	Huhtikuu 2012
Kieli	Englanti, suomenkielinen tiivistelmä
Sivumäärä	106 s. + liitt. 150 s.
Julkaisija	VTT PL 1000, 02044 VTT, Puh. 020 722 111

Global software engineering. Challenges and solutions framework

Globally distributed product development has become more and more common practice. In addition to own resources, companies need to employ resources on a global scale even from partner companies throughout the world, in order to produce software at a competitive level. In practice, the productivity in distributed software development drops up to 50% compared to single site software development. Main reasons behind this productivity drop are misunderstood or mismatched processes between teams, and poor visibility into and control of the development activities at all sites involved. The purpose of this thesis is to analyse in more detail why this is the case and what could be done to improve the situation in practice.

This thesis shows that although GSE is common, it is still challenging and companies should carefully weigh the benefits and costs of doing the work in distributed setting vs. doing it single site. This thesis is a step towards better, more productive and higher quality GSE, as it helps companies to be aware and address potential challenges early. Although the work is done in software intensive system development context, the results are largely applicable also for companies doing globally distributed development in other domains.

ISBN 978-951-38-7459-9 (soft back ed.)
ISBN 978-951-38-7460-5 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 2242-119X (soft back ed.)
ISSN 2242-1203 (URL: http://www.vtt.fi/publications/index.jsp)

VTT SCIENCE 6 Global software engineering. Challenges and solutions framework