VISIONS • SCIENCE • TECHNOLOGY • RESEARCH HIGHLIGHTS

Dissertation
70

Measurement-based management of global software development projects

Maarit Tihinen

VTT

VTT SCIENCE 70

Measurement-based management of global software development projects

Maarit Tihinen

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 28th of November, 2014, at 12 noon.

ISBN 978-951-38-8177-1 (Soft back ed.)
ISBN 978-951-38-8178-8 (URL: http://www.vtt.fi/publications/index.jsp)

VTT Science 70

ISSN-L 2242-119X
ISSN 2242-119X (Print)
ISSN 2242-1203 (Online)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT
PL 1000 (Tekniikantie 4 A, Espoo)
02044 VTT
Puh. 020 722 111, faksi 020 722 7001

VTT
PB 1000 (Teknikvägen 4 A, Esbo)
FI-02044 VTT
Tfn +358 20 722 111, telefax +358 20 722 7001

VTT Technical Research Centre of Finland
P.O. Box 1000 (Tekniikantie 4 A, Espoo)
FI-02044 VTT, Finland
Tel. +358 20 722 111, fax +358 20 722 7001

Grano Oy, Kuopio 2014

Preface

This research was carried out as part of several research projects at VTT Technical Research Centre of Finland. Many people have helped me during this long process 2003–2014. First, I mention my supervisor Professor Veikko Seppänen. Veikko became my thesis supervisor in the 2003, when there was some vague idea about a thesis related to measurements and metrics and their utilisation in software development, at both project and organisational levels. During the research process, the topic appeared to grow or even change, but measurements and metrics have been part of my research activities throughout the process. In fact, the research topic has matured into measurement-based management of global software development projects, and now the work is ready. I wish to thank Veikko for helping to define the scope of the thesis, for encouraging me during the writing, and for, sometimes, even providing a light push forward. I cannot thank him enough for all the guidance, valuable comments and support during the process. I wish also to thank my supervisor Markku Oivo for promoting my writing process with valuable discussions and for advising me to concentrate on the most important activities and topics. Without the support of my supervisors, this work would not have been possible.

The manuscript of this thesis was reviewed by Professor Hannu Jaakkola, of the Tampere University of Technology in Finland and Professor Sandro Morasca, of the University of Insubria in Italy. I deserve my sincere thanks for their time and effort they have spent in reviewing my research and giving their extremely constructive comments and recommendations, which have helped me to improve the quality of the thesis.

I am also grateful for VTT for giving me the opportunity to work on such interesting topics and complete my thesis in research projects. Over the years, I have worked with many great people, both at VTT and in the companies participating in the projects, and I offer many thanks to you all. This thesis is based on research carried out in collaboration with several participants acting together. The results were reported in six publications by key participants and the author of this thesis. I thank all co-authors for their contributions.

Without the industrial interest and involvement this research would not have been possible. I wish to give special thanks to Rob Kommeren from Philips and Jim Rotherham from Symbio for their active participation in workshops, case

studies and publications as well as for their valuable comments, ideas and feedback during the research.

My special thanks to my friend and workmate Dr. Päivi Parviainen for your friendship and encouragement throughout. You have helped and inspired me with discussions, innovations, and support and you have engaged my motivation to finalise this thesis. I would also like to mention Dr. Tuomo Tuikka who has encouraged me to finish this work and has allocated the time in which to do it. Furthermore, I would like to express my thanks to Tekes for funding the projects where this research has been carried out.

Finally, I wish to express my sincere gratefulness to my beloved ones for their loving support, understanding encouragement through the years: My mother-in-law, Raija, who has supported me in myriad domestic works without questions; my boys, Jorma and Henri, who have kept me in touch with reality with their tricks, delights, and love; and my husband, Vesa, who has supported me with altruistic love and patience. You have shown endless understanding of my evening and weekend working periods by unselfishly providing me the possibility to concentrate on the research. Thank you from the bottom of my heart.

Olhava, Finland, October 2014
Maarit Tihinen

Academic dissertation

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

Reviewers Professor Hannu Jaakkola
Tampere University of Technology
P.O. Box 300, FI-28101 Pori, Finland

Professor Sandro Morasca
University of Insubria
Department of Computer Science and Communication
Via Valleggio 11, I-22100 Como, Italy

Opponent Professor Tommi Mikkonen
Tampere University of Technology
P.O. Box 527, FI-33101 Tampere, Finland

List of publications

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

I Komi-Sirviö, S. and Tihinen, M. (Alphabetical order.) 2005. Lessons learned by participants of distributed software development. The Journal of Knowledge and Process Management. ISSN 1099-1441, Volume 12, Number 2, DOI: 10.1002/kpm.225. Pp. 108–122.

II Tihinen, M. and Järvinen, J. 2006. How to build and sustain a measurement data management environment in a SME. In: Proceedings of Software Measurement European Forum (SMEF). Rome, Italy. Pp. 225–236.

III Tihinen, M., Parviainen, P., Suomalainen, T., Karhu, K. and Mannevaara, M. 2011. ABB Experiences of Boosting Controlling and Monitoring Activities in Collaborative Production. In: Proceedings of the 6th IEEE International Conference on Global Software Engineering (ICGSE). Helsinki, Finland. Pp. 1–5.

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

V Tihinen, M., Parviainen, P., Kommeren, R. and Rotherham, J. 2011. Metrics in distributed product development. In: Proceedings of the 6th International Conference on Software Engineering Advances (ICSEA). Barcelona, Spain. Pp. 275–280.

VI Tihinen M., Parviainen P., Kommeren R. and Rotherham J. 2012. Metrics and Measurements in Global Software Development. The International Journal on Advances in Software. Volume 5, Number 3&4. Pp. 278–292.

Contents

Preface **3**

Academic dissertation **5**

List of publications **6**

1. Introduction **9**
1.1 Research questions and scope 10
1.2 Research design 11
1.2.1 Author's contribution to the research 15
1.2.2 Research process 16
1.3 Outline of the thesis 20

2. Related work **21**
2.1 Measurements and metrics 22
2.1.1 Approaches for measurements and metrics 22
2.1.2 Measurements automation 25
2.2 Global software development 26
2.3 Management of GSD projects 29
2.3.1 Management challenges 30
2.3.2 Knowledge-related challenges 31
2.3.3 Information needs for management 33
2.4 Measurements and metrics in GSD 35
2.4.1 Development-environment-related challenges 36
2.4.2 Metrics in the GSD literature 39
2.4.3 Measurements and metrics related challenges 39
2.5 Challenges in current measurements practices 42

3. Measurement-based management of GSD projects **46**
3.1 Dynamic measurements in GSD projects 47
3.1.1 Knowledge management and transfer in GSD 47
3.1.2 Requirements for dynamic measurements 49
3.2 Implementation of dynamic measurements in GSD projects 51
3.2.1 Technical viewpoints 52
3.2.2 Measurements viewpoints 55

3.2.3 Examples of industrial cases 59
3.3 Benefits of dynamic measurements 66

4. Original publications 72
4.1 Introduction 72
4.2 Paper I: Lessons learned by participants of distributed software development 75
4.3 Paper II: How to build and sustain a measurement data management environment in a SME 75
4.4 Paper III: ABB experiences of boosting controlling and monitoring activities in collaborative production 76
4.5 Paper IV: Knowledge related challenges and solutions in GSD 76
4.6 Paper V: Metrics in distributed product development 77
4.7 Paper VI: Metrics and measurements in global software development 78

5. Discussion 79
5.1 Validity of the research 79
5.2 Evaluation of the results 83
5.2.1 Theoretical contribution 84
5.2.2 Implications for the practice 86
5.3 Limitations of the research 88

6. Summary and conclusions 90
6.1 Summary of the results 90
6.2 Future research 91

References 94

Appendixes
Papers I–VI

1. Introduction

Today globally distributed product development is a common practice in industry because of its potential benefits, such as lower costs or taxes, the ability to respond to local customer's needs, and the possibilities to utilise resources globally. With distributed product development, either the development processes are distributed within an organisation (called multisite development) or two or more companies are involved in development activities. Although distribution brings its own special features for product development, products are increasingly more complicated, requiring multidisciplinary knowledge during the development processes. For example, trends in global software development show that the size and complexity of software-intensive systems will continue to grow. In spite of the potential benefits of global software development, several challenges and problems have been reported in task coordination, project management, and communication affected by distributed settings. Merely, the understanding of each other may not be straightforward, because of different backgrounds in the terms of terminologies and cultures (Komi-Sirviö and Tihinen 2003; Noll et al. 2010).

Managing a distributed product development project is more challenging than managing traditional development (da Silva et al. 2010). Based on an industrial survey (Komi-Sirviö and Tihinen 2003), one of the most important topics in project management associated with distributed software development is detailed project planning and control during the project, such as dividing work by sites into sub-projects and clearly defining responsibilities, dependencies, and timetables, along with regular meetings and status monitoring. In globally distributed development, cultural differences bring new challenges for management. For instance, there can be different viewpoints to problem reporting; at one site, it may be a regular, natural, and necessary action originating from a basic development procedure, whereas at another site, the required reporting can be understood as an insult for inducing active problem hiding. In addition, commitment to decisions and timetables can vary between stakeholders from different countries or continents.

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them to clarify defined rules (Fenton and Pfleeger 1998). It is generally understood that measurement and metrics are key technologies for managing and controlling product development projects. For example, the main purpose of both measurements and metrics in software production is to create the means for monitoring and controlling

which provide support for decision making and project management (Basili 1992). A successful measurement program can prove to be an effective tool for keeping on top of the development effort, especially for large distributed projects (Umarji and Shull 2009). However, many problems and challenges have been identified that reduce or even eliminate all interests in measurements. For example, not enough time is allocated for the measurement activities during a project or not enough visible benefits are gained for the project by doing the measurement work (e.g., data is useful only at the end of project, not during the project). Further, in many organisations, so called metric enthusiasts have produced and defined too many metrics, which have made collecting and analysing the data too time consuming.

Globally distributed development generates new challenges and difficulties in measurement practice results and metrics interpretation. For example, the gathering of the measurements data can be problematic because of various reasons, including the use of different development tools and their versions; work practices with related concepts, which can vary according to the project stakeholders; and the reliability of the gathered data, which can vary due to cultural differences, especially, in subjective evaluations. In addition, interpretation and decision making based on the measurement results require that the distributed development implications be taken carefully into consideration.

1.1 Research questions and scope

As global software development has huge potential benefits and possibilities, it is important to concentrate on avoiding identified problems and, thereby, to create new understanding, awareness, and means for successful production. The purposes of this thesis are to analyse, in detail, the effects of global software development and to research how project management practices within measurements and metrics can be improved. Research on global software development exists, but the viewpoint of measurement-based management has not been examined in its larger perspective. Only single company or case experiences and studies of specific aspects of project management practices or example metrics can be found. Thus, the research questions of this thesis are as follows:

Q1. What are the main measurements- and metrics-related challenges faced by companies when managing global software development projects?

Q2. What kinds of means and solutions can be found to respond to these identified challenges?

Q3. How can measurement-based management be implemented in global software development projects?

The first research question is aimed to clarify what kinds of measurements- and metrics-related challenges companies face during global software development, and how do measurements, metrics, and interpretations of measurement results

differ between a traditional 'single-site' development and a distributed development. The research question also makes clear how distribution affects measurements, measurements needs, as well as interpretations of measurement results in global software development (GSD).

The second research question is focused on tackling identified challenges while managing GSD projects. The aim is to clarify what kinds of measurements and metrics can be utilised for producing information for the management of GSD projects, and what kind of practical means or knowledge- related solutions for measurements and metrics should be considered in managing distributed software development projects. This research question is not only focused on potential metrics and measurements in GSD projects, instead the aim is also to clarify the features and requirements to determine the most useful measurement-based solutions for the management of GSD projects.

The third research question is posed to clarify how measurement-based management can be implemented in GSD. The main aim is to clarify the main requirements and the practical means for supporting measurement-based management and determining how those can be implemented; for example, to determine what kinds of features tools or selected reporting solutions should be involved in proposed management solution in GSD projects.

1.2 Research design

Creswell (2009) suggests that researchers should make explicit the larger philosophical ideas they espouse because this information helps to explain their decision to choose qualitative, quantitative, or mixed-methods approaches for their research. He introduced four worldviews that represent 'a basic set of beliefs that guide action'. The main elements of each worldview are presented in Table 1.

Table 1. The major elements of four worldviews, based on (Creswell 2009).

Post positivism	**Constructivism**
• Determination • Reductionism • Empirical observation and measurement • Theory verification	• Understanding • Multiple participant meanings • Social and historical construction • Theory generation
Advocacy/Participatory	**Pragmatism**
• Political • Empowerment issue-oriented • Collaborative • Change oriented	• Consequences of actions • Problem centred • Pluralistic • Real-world-practice oriented

The author espouses the constructivist worldview, even if some pragmatic approaches could be appropriated. Constructivists focus on the specific contexts

in which people live and work, to understand the settings of the participants. Researchers seek to understand the context or setting of the participants through visiting this content and gathering information personally. The researchers also recognise that their own backgrounds shape their interpretation, and they position themselves in the research to acknowledge how their interpretation flows from their personal, cultural, and historical experiences. The researchers develop subjective meanings of their experiences. The meanings are directed towards certain objects. These meanings are varied and multiple, leading the researcher to look for the complexity of views rather than narrowing meanings into fewer categories or ideas. The constructivism worldview utilises qualitative research methods (e.g., asking open-ended questions so that the participants can share their views). The process of qualitative research is typically inductive, whereas the researchers inductively develop a theory or a pattern of meaning (Creswell 2009; Järvinen 2012).

The author, to some extent, also considers a pragmatic worldview, especially, that research always occurs in social, historical, political, and other contexts. Pragmatism opens the door to multiple methods, different worldviews, and different assumptions, as well as different forms of data collection and analysis. The pragmatism worldview considers that individual researchers have the freedom of choice: they are free to choose the methods, techniques, and procedures of research that best meet their needs and purposes. The pragmatism worldview applies to mixed-methods research, in that the researchers draw liberally from both quantitative and qualitative assumptions when they engage in their research (Creswell 2009).

The structured taxonomy (Järvinen 2012) can be utilised while identifying, planning, or selecting the research approach and methods. Järvinen (2012) enumerates research strategies using mathematical approaches, conceptual-analytical approaches, theory-testing and theory-creating approaches, and the building and evaluation of innovations. The taxonomy of different research approaches is presented in Figure 1.

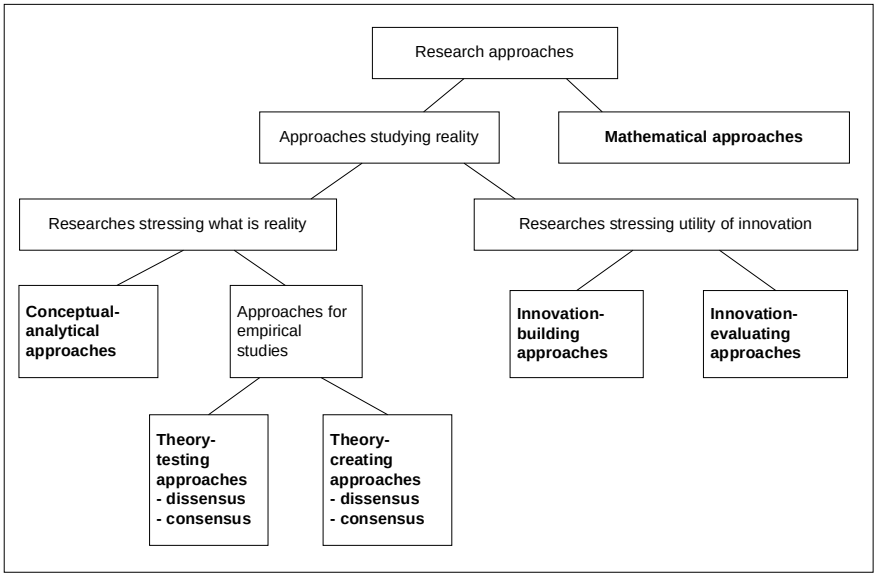

Figure 1. Järvinen and Järvinen's taxonomy, based on (Järvinen 2012).

The research of this thesis can be classified as applied research, and more specifically, as constructive research. Järvinen (2012) introduces 'constructive research' as involving the building and evaluating of innovation based on existing knowledge (research) and new technical or organisational advancements. Furthermore, Järvinen suggests that in constructive research, it is possible to accept a prototype or even a plan as a research outcome instead of a final product. This approach was utilised during the research process because there is no only one exact solution that is appropriate for all situations in GSD. Thus, a proof of concept tool integration solution was developed to enable the study of measurements and metrics in different collaborative settings (e.g., industrial experiences, such as the challenges and benefits). In addition, the design science approach was emphasised for building and evaluating the innovation. The design science is focused on the utility of the innovation: the new artefact is built for identified business needs. In fact, the principles of design science bring into sharp focus knowledge and understanding of design problems and their solutions in the building and application of artefacts (Hevner and Chatterjee 2010).

Yin (2009) introduces three main conditions: 1) the type of research questions, 2) the extent of control an investigator has, during actual behavioural events, and 3) the degree of focus on contemporary as opposed to historical events. In addition, Yin (2009) discusses how each is related to the five major research methods: experiments, surveys, archival analyses, histories, and case studies. The importance of each condition in distinguishing amongst the five methods is presented in Table 2.

Table 2. Relevant situations for different research methods, based on (Yin 2009).

METHOD	(1) Form of research questions	(2) Does it require control of behavioural events?	(3) Does it focus on contemporary events?
Experiment	How, why?	Yes	Yes
Survey	Who, what, where, how many, how much?	No	Yes
Archival analysis	Who, what, where, how many, how much?	No	Yes /no
History	How, why?	No	No
Case study	How, why?	No	Yes

This study utilises case study research to build and evaluate the research outcomes of the thesis. As shown before (Table 2), case studies are the most useful approach to answer 'why' and 'how' questions in situations where controls over behavioural events are not required, and where the focus is on contemporary events (Yin 2009). Furthermore, during the research process of this thesis, several literature studies were conducted for creating an appropriate, extensive, and well-focused theoretical framework of the research topics introduced in this thesis. Because the actual research was performed during several years (in fact, over ten years), within several research projects, it was possible to construct the achieved results of case studies based on existing knowledge and research. Thus, the research process involved several literature studies focused on the most relevant topics to the research questions.

In this thesis, various single-case studies were investigated. The data and information gathering methods were interviews, observations, e-mail inquiries, and becoming familiar with the documentation, databases, tools, and Intranet of the case study companies. Järvinen (2012) emphasises the possibilities of case studies to examine very complicated circumstances and, in this way, to gather new information for creating new knowledge. According to Yin (2009), the single-case design is eminently justifiable if the case represents (a) a critical test of existing theory, (b) a rare or unique circumstance, or (c) a representative or typical case, where the case serves a revelatory or longitudinal purpose. The author of the thesis emphasises that the environments and circumstances of measurements and metrics in each organisation are unique depending on cultural, historical, and technical issues and backgrounds, production processes, or collaboration modes, for example. Thus, the use of case research was selected to provide experiences and to gather new information and gain understanding of complicated circumstances. Case studies are especially appropriate when the context is expected to play a role in the phenomena (e.g., if the stresses of a real project affect developers' behaviour), or when effects are expected to be wide ranging or are expected to take a long time (e.g., weeks, months, or years) to appear (Easterbrook et al.

2008), which was the case in the present research. Further, in design science research, a designer answers questions that are relevant to human problems via the creation of innovative artefacts. Therefore, the designed artefacts are both useful and fundamental in understanding those identified problems and in contributing new knowledge to the body of scientific evidence (Hevner and Chatterjee 2010).

1.2.1 Author's contribution to the research

Since 2000, the author has participated in several research and customer projects where it has been possible to research topics – measurements and metrics, knowledge management, project management, and GSD – in greater detail, not only based on literature studies but also in industrial environments.

The author was as a research scientist in the MIKKO project during 2000–2001 (MIKKO project – Comprehensive collection and utilisation of software measurement data) funded by Tekes, the Finnish Funding Agency for Innovation (www.tekes.fi/en). The main goal of the project was to develop a comprehensive measurement framework for the industrial software processes to support measurement data collection and utilisation at the project and organisational level. During the research project, the author participated in few case studies, co-authored conference papers, and participated in producing the MIKKO Handbook (Vierimaa et al. 2001), the main result of the project. In addition, the author did her secondary subject thesis for the University of Oulu, titled 'Analysis of the Quality Measurement Processes in Software Production' (Tihinen 2001). The MIKKO research project included large literature studies that built the firm's basis for further research work with measurements and metrics and in environments of GSD.

During 2001–2003, the author participated as a project manager in the Knots-Q project funded by the Finnish Academy (Knots-Q project – Knowledge-centered tools and methods for software process quality improvement, http://virtual.vtt.fi/virtual/proj1/projects/knots-q/index.htm). The goal of the project was to develop knowledge-centred tools and methods for improving the quality of software production. The author's research interests expanded to knowledge management aspects. The Knots-Q project enabled an industrial survey of distributed software development (Komi-Sirviö and Tihinen 2003), for gathering information and constructing new knowledge of GSD. The author contributed to creating the questionnaire as well as gathering the responses and analysing the work. The survey exposed the most serious problems faced in distributed software development. In addition, the survey elicited practical knowledge and examples of the problems experienced along with the potential solutions that have been developed and tested by developers. The research not only enriched the author's skills of distributed software development but also led to consideration of a knowledge engineering approached, both in global software development as well as in practices of measurement and metrics of GSD projects.

During 2004–2007, the author continued the research in the ITEA MERLIN project (MERLIN project – Embedded Systems Engineering in Collaboration, http://virtual.vtt.fi/virtual/proj1/projects/merlin/), which focused on embedded systems engineering and software engineering technologies from a collaboration perspective. During the project, the author contributed actively with other research scientists and partners in producing the MERLIN Collaboration Handbook (Parviainen et al. 2008), which was the main result of the project. The Internet-based handbook provides concrete solutions to support collaboration in practice; solutions were collected from a large number of experiences from industrial companies. In addition, literature resources were included to guarantee a solid methodological basis for decision making. The MERLIN Collaboration Handbook covers various collaboration modes and describes solutions to support success in the critical activities of collaborative product development project. The author participated in literature studies as well as industrial cases during the MERLIN project. For example, the first construct of a measurement-data management framework was created and studied via the case environment.

The ITEA2 project PRISMA project (PRISMA – Productivity in Collaborative Systems Development, https://itea3.org/project/prisma.html) 2008–2011 was focused on improving the productivity of collaborative systems development through improved development technologies. The author was responsible for research actions according to the project plan (e.g., participating in several case studies that built a basis for the thesis). In addition, the author was responsible for packaging results by leading the dissemination work package of the project. All project results were packaged to be available from the Intranet (www.SameRoomSpirit.org). The SameRoomSpirit Wiki was based on the MERLIN Collaboration Handbook developed during the MERLIN project. This wiki-based handbook was updated and enhanced based on literature studies and industrial case studies processed during the PRISMA project.

1.2.2 Research process

In this subsection, the research process and the most relevant research outcomes in relation to the thesis are summarised. As mentioned before, the author has actively participated as a research scientist in several research projects. Figure 2 depicts how those research projects were chronologically scheduled and what kinds of research results were produced. In Figure 2, the horizontal axis shows time and how the three main phases of the research process can be distinguished. The vertical axis depicts how each main phase includes the three dimensions of the research: 1) literature studies with focused industrial inventories or surveys, 2) actual case studies, and 3) main research results based on analyses and conclusions.

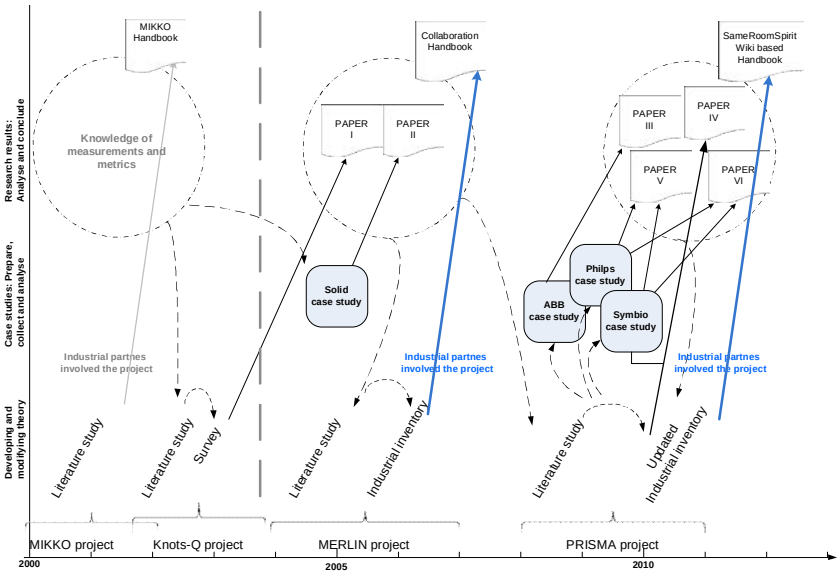

Figure 2. Overview of the research process.

The first phase included the MIKKO and Knots-Q research projects from 2000 to 2003. This phase provided a theoretical background for measurements and metrics and practices of distributed software development, as well as a base for further research activities. The second phase included the MERLIN research project (2004 to 2007) with a literature study and an industrial inventory focusing on collaboration issues during software development. In this phase, the first case study and two original papers (Paper I and Paper II) were finalised and published. In the third phase, the PRISMA project (2008–2011) was carried out. This phase included a more detailed literature study, an updated industrial inventory, case studies focused on the research questions of the thesis, and the four original publications (Paper III, IV, V and VI).

As shown in Figure 2, literature studies were carried out in several phases. The first literature study was done in 2000 during the MIKKO project. The purpose of the study was to gather and become familiar with the existing research of measurements, metrics, and process improvement practices used during software production. Even if this literature study was partially utilised in producing the MIKKO Handbook, the author's goal was to build a theoretical background for her secondary subject thesis. These research results together built a strong theoretical base for software measurement processes and metrics. Based on the research, a reference model – a measurement-data management framework – was built, and its applicability in a case company was studied (Paper II). Further, from the viewpoint of a project manager, it is important to understand the circumstances of the projects being managed. Thus, knowledge-management practices for

improving the quality of software production were taken into consideration during the Knots-Q project (2001–2003). Meanwhile, it was also recognised that distributed development was increasingly common product-development practice. Therefore, a second literature study was performed to focus on gathering other researchers' views on and experimental results of distributed SW production. This literature research built the theoretical background that was utilised while the survey (Paper I) was being planned and carried out. Hence, the research context of the thesis was specified: measurements, metrics, project management, and GSD.

The next literature study was carried out in 2004, at the beginning of the MERLIN project. The MERLIN project focused on production technologies and methods from a collaboration perspective. The main goal of the study was to gain knowledge of collaborative work and practices that is more detailed (e.g., what kind of collaboration modes can be identified, what are the most problematic or critical issues relating to collaborative work, and what are the most important areas to which research activities should be directed. Based on the study, a framework was defined to collect information during an industrial inventory. The author participated in both studies, focusing on topics of project management, quality management, and measurement and metrics, amongst others. The research revealed information of GSD, in greater detail; for example, how different collaboration modes – customer-supplier relationship, technology exchange /licensing, joint research and development, or in-house distributed development (Parviainen 2012) – affect project management. Based on the literature study and the industrial inventory, the most important challenges for GSD were identified. Then solutions for these challenges were sought and tested in industrial practice. These challenges and solutions were collected for the MERLIN Collaboration Handbook (Parviainen et al. 2008), the main result of the project. Thus, during the MERLIN project, the author was able to connect the produced research results to her earlier research results (e.g., the MIKKO Handbook, Paper I, and Paper II). The author understood that project management plays a central and very important role during GSD that needs to be supported by all possible means. In GSD, production processes vary greatly from project to project depending on factors such as the product, collaboration modes, or the number of stakeholder involved. Because traditional software-process-improvement actions lose ground, metrics and measurements shall be focused on supporting management practices.

The next literature study was carried out in 2009, during the PRISMA project. Because of the literature and industrial studies that were done in the earlier MERLIN project, this literature study could be focused on searching for new concrete solutions from experiments in GSD. In addition, a workshop with ten industrial partners was arranged to identify new challenges that the companies were facing in their GSD projects as well as their proposed solutions. Thus, the research could be focused on those topics for which solutions were not be found or reported. Then new potential solutions were sought, and the proposed solutions were tested in industrial circumstances (e.g., Paper III, Paper V, and Paper VI). In addition, because the survey (Paper I) highlighted the crucial role of knowledge and knowledge engineering in managing GSD projects, the research of knowledge-

related challenges and knowledge needs from the viewpoint of different stakeholders in global software development were also further examined (Paper IV) during the PRISMA project. In practice, all collected, developed, or tested challenges and solutions were included in the second version of the Collaboration Handbook. The research process and results are introduced and discussed in more detail in the dissertation (Parviainen 2012). Because of the large amount of information involved, the challenges and solutions were also documented as a wiki solution (www.SameRoomSpirit.org), being freely available from the Internet.

Figure 3 summarises the research outputs and depicts how the research was adjusted piece by piece for finding answers to research questions.

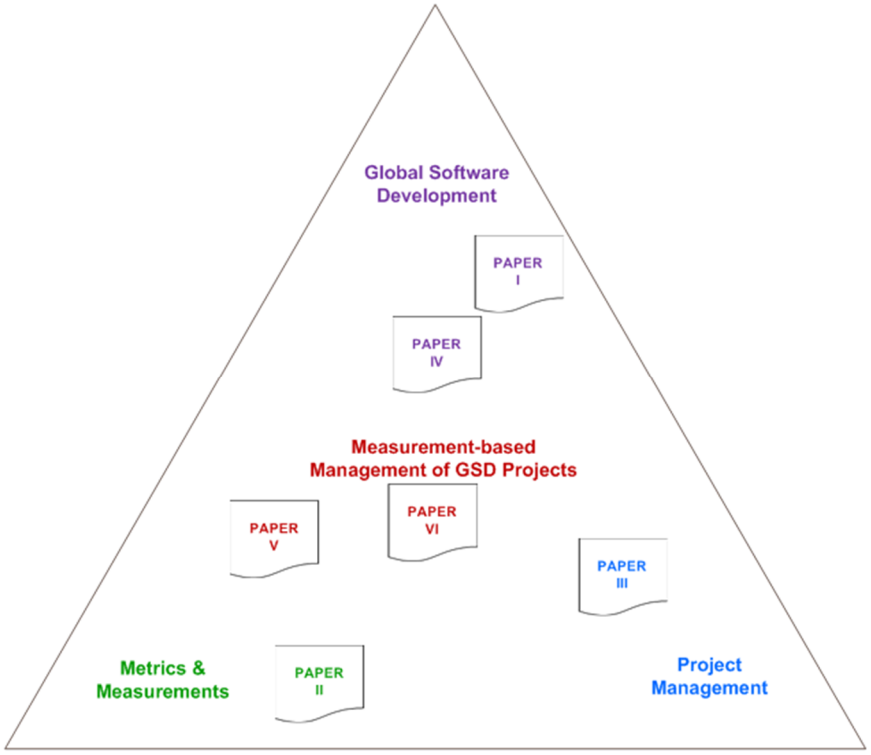

Figure 3. Summary of the research process.

The main new contributions of the thesis include the research results in the context of measurement-based management of GSD projects.

1.3 Outline of the thesis

The structure of the thesis is presented as follows:

- Section 1 introduces the background and focus of the thesis, with research questions and the author's contribution to the research projects where the research results have been processed and validated. Moreover, the research design with a description of the research process is presented.
- In Section 2, the related work of the thesis is discussed. The related work consists of the main research topics: 1) measurements and metrics in general, 2) GSD, 3) management of GSD projects, and 4) measurements and metrics in GSD. Also, the identified challenges faced in the current measurements practices are summarised.
- Section 3 analyses the research results provided via industrial case studies and literature studies during the research projects. In addition, a definition of *dynamic measurements* is introduced and the concept of measurement-based management of GSD projects is discussed.
- Section 4 introduces the original publications that the thesis summarises, and how each publication answers or clarifies the research questions of the thesis. Also, author's contribution to each publication is clarified.
- Section 5 discusses the research results, including evaluation of the results with respect to the research goals and validity of the research. Moreover, the limitations of the research are discussed.
- Section 6 summarises and concludes the research results, and offers insights for future research.

Even if each section provides the main content for the thesis, a logical interrelationship among the sections is clear.

2. Related work

GSD is shortly defined as software development that uses teams from multiple geographic locations. According to (Sangwan et al. 2006), GSD can include teams from the same organisation or teams from different organisations that are involved by collaboration or outsourcing, for example. They point out that the distance of teams is an essential factor when teams are working together to develop software systems. Thus, GSD can even consider software development where teams are within one country, if the development teams are separated by more than 50 metres (Sangwan et al. 2006).

GSD has become the norm in software-intensive systems development. In practice, the products are increasingly developed globally, in collaboration between subcontractors, third-party suppliers and in-house developers (Hyysalo et al. 2006; Noll et al. 2010). Project control and management activities are now increasingly important, as software products are developed in dynamic environments where requirements, priorities, participating sites, development processes and tools, and even partners are continuously changing. In fact, up-to-date information of project status is now critical to completing project management activities effectively.

In this section, the research relevant to the thesis is introduced. The related work includes the results of the literature studies and also results of large industrial inventories carried out during the MERLIN and PRISMA projects. The results of inventories were included as they enabled the author to collect a large set of empirical data about industrial challenges as well as views of potential solutions relating to measurements and project management in GSD. This empirical work was carried out over several years. The first industrial inventory was made during the MERLIN project, included also industrial experiences reported in the literature. During the PRISMA project, several industrial workshops were held to define challenges and potential solutions for these challenges. These industrial inventories not only provided empirical data to the author but also assisted to focusing the literature studies in the context of the thesis.

2.1 Measurements and metrics

The main purpose of measurements and metrics in software production is to create the means for monitoring and controlling that provide support for decision making and project management (Basili 1992). In this subsection, the works related to measurements and metrics are introduced.

2.1.1 Approaches for measurements and metrics

Since the 1960s, software measurements and metrics have been discussed. Software metrics are a valuable factor for the management and control of many software-related activities; for example, costs, effort and schedule estimation, productivity, reliability, and quality measures. Traditionally software measurement has been understood as an information gathering process. Software measurement is defined (van Solingen and Berghout 1999) as *'the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products*'. Further, traditionally, software metrics are divided into process, product, and resource metrics (Fenton and Pfleeger 1998). In a comprehensive measurement program, all of these dimensions should be taken into consideration while interpreting measurement results; otherwise, the interpretation may lead to wrong decisions or incorrect actions. A successful measurement program can prove to be an effective tool for keeping on top of the development effort, especially for large distributed projects (Umarji and Shull 2009).

The measurement data items consist of numeric data (e.g., efforts and schedules) or a pre-classified set of categories (e.g., severity of defects: minor, medium, or major). Software metrics can consist of several measurement data items, singly or in combination. In fact, the literature often considers several kinds of static measures, such as lines of code, cohesion, or coupling. Instead, (Lavazza et al. 2012) proposed measures of dynamic internal software attributes and provided examples to show how dynamic measures can be defined, collected, and used. The dynamic measures provide new approaches.

Metric visualisation is a visual representation of the collected and processed information about software systems. Typically, software metrics are visualised for presenting the information in a meaningful way that can be understood quickly. For example, visualising metrics through charts or graphs is usually easier to understand than are long textual or numerical descriptions. Thus, metrics visualisation is an important task in a measurement process. Tufte and Graves-Morris (1983) emphasise that excellence in statistical graphics consists of complex ideas communicated with clarity, precision, and efficiency. Furthermore, they introduce that graphical displays should:

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else
- avoid distorting what the data have to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a data set.

Tufte and Graves-Morris (1983) advice that often the most effective way to describe, explore, and summarise a set of numbers is to look at pictures of those numbers.

In addition to data items, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparisons should be done within the same kind of development projects, or at the least differences should be taken into account. Traditional project characteristics are, for example, size and duration of a project, type of a project (development, maintenance, operational lifetime, etc.), project position (contractor, subcontractor, internal development, etc.), type of software (hardware-related software development, application software, etc.), or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model, etc.). Further, different phases of development projects have to be taken into consideration while analysing the gathered measurement data.

In the literature, many potential approaches for supporting measurements activities have been introduced, from their planning to taking into daily practices. For example, the IEEE standard for a software-quality-metrics methodology is a systematic approach to establishing quality requirements and identifying, implementing, analysing, and validating the process and product of software quality metrics for a software system (IEEE Std 1061-1992). The software-quality-metrics methodology can be used in all software at all phases of the software life cycle. The methodology applies to those associated with the acquisition, development, use, support, maintenance, and audit of software as well as most of the people who measure and assess the quality of the software. The methodology introduced has five main steps: 1) establish software quality requirements, 2) identify software quality metrics, 3) implement the software quality metrics, 4) analyse the software quality metric results, and 5) validate the software quality metrics. In addition, the IEEE standard defines factors and associated sub-factors for software quality. The factor metrics approach is a practical way to define

metrics because there are many examples associated with the factor and sub-factor metrics in the literature.

One of the most commonly used measurement methods at the end of 1990 and the beginning of 2000 was the Goal/Question/Metric (GQM) method. The GQM paradigm (Basili 1992) represents a systematic approach for tailoring and integrating the objectives of an organisation into measurement goals and their step-by-step refinement into measurable values. The GQM method was commonly known and was often used for searching and identifying organisations' strengths and weaknesses relating to the identified improvement goals. Further, several assessment methods, for example CMMI (2006) and SPICE (Software Process Improvement and Capability Determination, further known as a standard ISO/IEC 15504 Information technology – Process assessment), were generally used for identifying possible improvements areas and gaining knowledge of the software process of an organisation. In fact, most of the traditional measurement methods are based on expressions of the famous Shewhart cycle, also called the Deming cycle: PDCA (Plan–Do–Check–Act) (Shewhart 1939). The PDCA cycle is an iterative four-step management method that is used in business for the control and continuous improvement of processes and products. The traditional methods used in software measurements were generally based on clearly defined and largely stabile processes that could be adjusted and improved. In those cases, the improvement actions were mainly done afterwards, for example, in the next project.

A balanced score card (BSC) is widely used for monitoring the performance of an organisation towards strategic goals. The original BSC approach covers a small number of performance metrics from four perspectives, called Kaplan & Norton perspectives: Financial, Customer, Internal Processes, Learning & Growth (Kaplan and Norton 1992). The BSC framework added strategic nonfinancial performance measures to traditional financial metrics to give managers and executives a more 'balanced' view of organisational performance. However, many early BSCs failed because clear information and knowledge about the selection of measures and targets were not available. For example, organisations had attempted to use Kaplan & Norton perspectives without thinking about whether they were suitable in their situation. Many improvements and enhancements have been made to the BSC approach, and, since 2000, it has been described as a 'Third Generation' of balanced scorecard designs. The BSC has evolved to be a strategic management tool that involves a wide range of managers in the strategic management process. In addition, it provides boundaries of control but is not prescriptive or constrictive, and, more importantly, it removes the separation between the formulation and implementation of strategy (Lawrie and Cobbold 2004). The BSC suggests that the organisation should be viewed from four perspectives (learning & growth perspective, business process perspective, customer perspective, and financial perspective) and that metrics should be developed and data should be collected and analysed in relation to these perspectives. Even if BSC are generally intended to deal with strategic issues, the balancing of various perspectives of BSC has also been emphasised in

measurements. In fact, it has been proven that practical software measurement and the balanced scorecard are both compatible and complementary (Card 2003). Especially in the GSD context, in making decisions or taking actions based on the analysis of metrics and measurements collected from different development parties or stakeholders need to consider the specific GSD factors as well.

2.1.2 Measurements automation

Komi-Sirviö et al. (2001) define *measurement automation* as a means to automated data collection, calculation, and formation of analysis graphs repeatedly in a predefined way, using data stored in one or several data sources. In addition, Batagelj et al. (2008) introduced the prerequisites for measurement automation, such as measurement instruments with communication interfaces, computers with matching communication interfaces, and software development platforms for the measurement automation. Accordingly, in the literature, there are many reported approaches for solutions for measurement automation, ranging from simple data-acquisition applications to complex laboratory information systems, which combine data acquisition, data processing, data presentation, and data management. For example, Komi-Sirviö et al. (2001) found that measurement automation support could be a feature of a tool where the data originally exists or it could be a separate software package that executes the measurement automation tasks. Batagelj et al. (2008) found that measurement automation software must be as follows: 1) reliable and robust, 2) user-friendly and easy to use, without ambiguous settings and parameters, 3) properly validated, 4) equipped with instructions for use and/or built-in help, 5) protected against intrusion in data and program code, and 6) provided with means for the detection and correction of procedural and measurement errors.

Komi-Sirviö et al. (2001) pointed out that measurement automation, with proper tool support, is a key success factor to effectively managing projects in a large, multisite software-development environment. In addition, measurement automation can also be used to enhance the visibility of the measurement process, which can lead to a greater awareness of the reasoning behind collecting measurement data and utilising it within an organisation (Komi-Sirviö et al. 2001). According to Batagelj et al. (2008), the benefits of measurement automation include reduced workload, improved insight in the measurement process and data processing, and increased reliability of measurement results because of reduction of human errors. However, it has been determined that automated systems for measurement and analysis are not adopted widely in companies despite the opportunities they offer (Coman et al. 2009). From a measurements viewpoint, only data collected from different sources meet the essential but demanding requirements for technical implementation. In practice, measurement activities, such as planning the measurements and packaging the experiences, fall outside the measurement-automation context because they depend on human judgement (Komi-Sirviö et al. 2001). Hence, measurement automation is not a

comprehensive solution for the challenges of measurements and project management. As Buse and Zimmermann (2010) argued, the information currently delivered by existing tools does not match the needs of project managers. The information is not valid for making good decisions. Measurement automation is not the answer to these problems; other practices are needed. For example, information and support needs in GSD have to be studied and the weaknesses of current practices need to be reviewed.

2.2 Global software development

Trends in GSD 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 (Forrester 2010; Fryer and Gothe 2008). Thus, the products are being increasingly developed in a globally distributed fashion (Hyysalo et al. 2006; Noll et al. 2010).

In practice, GSD enables many advantages that organisations are seeking while selecting strategies for distributed software development (Lings et al. 2007). It has created opportunities for companies to distribute their software development, for example, to economically favourable countries, to gain needed expertise from top universities, or to get closer to customers. The collaboration also offers an opportunity to leverage time-zone differences, or organisations are able to keep their key workers who had chosen to move, for example. Based on industrial inventory (including 12 interviews in five companies and two questionnaires filled in two companies) made during the MERLIN project, the most common motivation for collaboration, mentioned by almost all of the inventoried companies, was to save money. The next most commonly mentioned reasons were to acquire competence (technology competence or knowledge of a certain market) that is not available in house, and to avoid investing in the company's non-core competence areas. Other reasons for collaboration were to save time, to establish new business opportunities with new partners, to have flexibility with respect to the number and the availability of in-house resources. In some cases (e.g., COTS (Commercial-Off-The-Shelf), the developer is a pure subcontracting or consulting company, and the whole business is based on collaboration.

In practice, distributed projects struggle with the same problems as single-site projects, including problems related to managing quality, schedules, and costs, and distribution makes it more difficult to handle and control these problems (Herbsleb et al. 2000; Herbsleb 2007; Jiménez et al. 2009). The actual challenges can be caused from various issues; for example, the distance between partners reduces communication, causing wrong assumptions and misunderstandings or differences in the background knowledge and skills of the partners can lead to wrong conceptions of the task/problem involved.

Parviainen (2012) describes problems and challenges that are directly caused by the basic GSD circumstances. These challenges influence measurements and metrics and their interpretation during GSD. These challenges are mainly an

intrinsic and natural part of GSD, and they can either complicate distributed product development or cause further challenges. The basic GSD circumstances are:

- Multiple parties, meaning two or more different teams and sites (locations) of a company or different companies
- Time difference and distance that are caused by the geographical distribution of the parties or teams.

Problems caused by these circumstances centre on issues of clarity of, for example, about roles and responsibilities for the different stakeholders in different parties or locations, knowing the contact persons (e.g., responsibilities, authorities, and knowledge) from different locations, and establishing and ensuring a common understanding across distance. The basic GSD circumstances can also lead to poor transparency and control of remote activities as well as difficulties in managing dependencies over distance, problems in coordination and control of the distributed work, and integration problems, for example. Problems may also be caused by basic circumstances in terms of accessing remote databases and tools or, accordingly, 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. The basic circumstances may also cause problems with data security and access to databases or another organisation's resources.

A commonly referenced classification for the challenges caused by GSD is as follows (Carmel 1999; Carmel and Tija 2005):

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

Communication breakdown (loss of communication richness). Human beings communicate best when they are communicating face to face. In GSD, face-to-face communication decreases because of distance. Minor or decreased communication causes misunderstandings and a lack of information regarding the sites or stakeholders involved in the project. For example, there can be information that is self-explanatory for the most of project members but is never known by others, and, on the other hand, communication over great distances (and cross-culturally) can lead to misinterpretations because people cannot communicate well due to language barriers. In addition, there are many differences in how different cultures use speech and communicate at meetings, listen to others, oversee their subordinates, or manage their staff (Lewis 1999). These kinds of differences may cause communication breakdowns, as even silence can be misunderstood depending on one's cultural background.

Coordination breakdown. Software development is a complex process 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. This can cause problem solving to be delayed or the project to go down the wrong track until it becomes very expensive to correct the course. GSD 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 needs to be defined (Damian and Zowghi 2003; Herbsleb and Mockus 2003; Paasivaara and Lassenius 2003). 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 the large number of independent teams. Wahyudin et al. (2007) stated that GSD demands more from project management. In addition to the project managers, the project members such as testers, technical leaders, and developers also need to be kept informed and notified of certain information and events that are relevant to their roles' objectives in timely manner, which provides the conditions for in-time decision making.

Control breakdown (geographical dispersion). GSD 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) must be used. These types of communication tools could be consider as less effective, not always providing the clear and correct status of the development site. Moreover, dividing the tasks and work across development sites and managing the dependencies between sites is difficult because of the restraints of the available resources, the level of expertise, and the infrastructure (Battin et al. 2001; Herbsleb and Moitra 2001; Welborn and Kasten 2003). According to Holmstrom et al. (2006), creating the overlap in time between different sites is challenging, despite the flexible working hours and communication technologies that enable asynchronous communication. Lack of overlap leads to a delay in responses with feelings 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 the tight social bonds that are a key to the project's success. Lack of informal communication and the use of different processes and practices have a negative impact on teamness (Battin et al. 2001; Damian and Zowghi 2003; Herbsleb and Mockus 2003). Further, fear (e.g., of losing one's job to the other site) has direct negative impact on trust, team building co-operation, and knowledge transfer, even where good relationships existed beforehand. According to Casey and Richardson (2008), fear and lack of trust negatively affect 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) discussed 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 could 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, which leads to a more effective collaboration.

Culture clash (cultural differences). Each culture has different communication norms. In any cross-cultural communication, the receiver is more likely to

misinterpret messages or cues. Hence, miscommunication across cultures is usually present. Hofstede (1984) argued that national and regional cultural groups influence the behaviour of societies and organisations. He introduced four dimensions of national culture through which cultural values and behaviour can be analysed: individualism-collectivism, uncertainly avoidance, power distance (strength of social hierarchy), and masculinity-femininity (task orientation versus person orientation). Borchers (2003) used Hofstede's model as a framework to study how cultural differences affected the software engineering techniques used in the analysed case projects. The results showed that the cultural indexes – power distance (e.g., degree of inequality of managers versus subordinates), uncertainty avoidance (e.g., tolerance for uncertainty about the future), and individualism (like the strength of the relationship between an individual and their societal group) – were found to be relevant from the software engineering point of view (Borchers 2003). Furthermore, Lewis (2006) stated that the national and regional cultures of the world can be generally classified into three groups: linear-active, multi-active, and reactive. These classifications take individual characteristics into consideration, such as being task or people oriented, highly or loosely organised, introverted or extroverted, time or task oriented, and confrontational or reserved. The Lewis model of cross-cultural communications provides a practical framework for understanding and communicating with people of other cultures. For example, Holmstrom et al. (2006) discussed the challenge of creating a mutual understanding between people from different backgrounds. They concluded that often, general understanding in terms of English was good, but more subtle issues, such as political or religious values, caused misunderstandings and conflicts during projects.

2.3 Management of GSD projects

Project control and management activities and abilities are highly important when the products are developed globally in distributed manners, such as in collaboration between subcontractors, third-party suppliers (open source, COTS, components, etc.), or in-house developers. It is commonly understood that project management covers activities such as planning, scheduling, organising, controlling, and managing tasks and resources to achieve the successful completion of a set of specific project goals. For example, Lotlikar et al. (2008) highlighted the importance of tracking and coordination in their definition of project management as the tracking and coordinating of tasks in a project so that the entire project is completed on time and within budget.

The Standish Group research (CHAOS Report 2014) shows that about 31.1% of projects will be cancelled before they are ever completed, and 52.7% of projects will cost 189% of their original estimates. The sample included large, medium, and small companies across major industry segments (e.g., banking, securities, manufacturing, retail, wholesale, health care, insurance, services, and local, state, and federal organisations). The cost of these failures and overruns are not

measureable but could easily be in the trillions of dollars. Thus, it is important to research in detail why projects fail or succeed. Based on the Standish Group research, the three major reasons that a project will succeed are 1) user involvement (15.9% of responses), 2) executing management support (13.9%), and 3) clear statement of requirements (13.0%). There are also other success criteria identified in the research, but with these elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically.
In GSD, a project manager may be far away from the development teams, 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 because of the decreased visibility. That is why, in distributed projects, the systematic controlling and status reporting of the project work is especially important, and, in fact, measurement and metrics provide important means to do that effectively.

2.3.1 Management challenges

Managing a distributed product development project is more challenging than managing a traditional development project (da Silva et al. 2010). In fact, project management and coordination mechanisms are largely emphasised as means for avoiding project failure in GSD. For example, Herbsleb (2007) argued that the key phenomenon of GSD is coordination over distance: the need to manage a variety of dependencies across sites drives the essential problems of GSD. Accordingly, Wu et al. (2009) argued that project management is one of the primary factors to a software project's success or failure. They noted that software projects often fail because the mangers do not know the true project status. Project managers need to get real-time information of project progress for performing various kinds of analysis on project data and for making decisions based on the analysis. In addition, software project planning is one of the most critical activities in the project management process. Without a realistic and objective software project plan, the software development process cannot be managed in an effective way (Wu et al. 2009). In GSD, the planning includes (e.g., dividing work by sites into subprojects, clearly defined responsibilities, dependencies, and timetables, along with regular meetings and status monitoring). The same kind of management challenges were identified in our survey (Paper I) that focused on the most serious problems faced in distributed software development. Paper I concludes that successful distributed software development requires both structured – and disciplined – software engineering and knowledge-management solutions embodying, most particularly, communication management and the utilisation of effective substitutes for face-to-face communication. Moreover, Jones (2004) introduces the congruent results via the analysis of about 250 large software projects that were examined between 1995 and 2004. Table 3 summarises six major factors noted at opposite ends of the spectrum in terms of failure versus success, as they were revealed in Jones's study analysis.

Table 3. Major factors for successful and failing projects, based on (Jones 2004).

Successful projects	Failing projects
Effective project planning	Inadequate project planning
Effective project cost estimating	Inadequate cost estimating
Effective project measurements	Inadequate measurements
Effective project milestone tracking	Inadequate milestone tracking
Effective project change management	Inadequate change management
Effective project quality control	Inadequate quality control

Jones (2004) argues that project management is the factor that tends to push projects along either the path of success or the path of failure. The critical activities for successful projects are project's planning, estimating, change control, and quality control. Thus, the most important software development practices leading to success are those of planning and estimating before the project starts, absorbing changing requirements during the project, and successfully minimising bugs or defects. However, the management and control of GSD projects is very demanding and complicated because there can be limited availability or no access at all to the needed project data.

2.3.2 Knowledge-related challenges

In global product-development projects, collaboration between persons, teams, various roles, sites, and organisations may result in new challenges that need to be managed during development projects. During the MERLIN project, industrial partners were interviewed to identify the management problems they faced in their own collaborative development projects. The following are brief descriptions of the kinds of challenges and problems they faced in practice:

- Some managers find it difficult to establish an effective relationship with the staff of other organisations. Clearly, the project manager is not able simply to issue instructions. The most common failing, however, seems to be hesitant management that leaves team members unclear about what is expected of them.
- Collaboration usually means that the project is split across two or more sites, possibly a considerable distance apart. This makes communication, particularly informal communication and the quick identification and resolution of problems, more difficult.
- The different organisations may well have different or even conflicting standards and work practices that must be resolved at an early stage.

- At a deeper level, every organisation has its own culture, reflected in work patterns, the way the organisation interacts with the outside world, and the way authority is distributed, and the general level of competence-collaboration involves adapting to the culture of one or more other organisations.
- The complications of multiorganisation management, communications, standards, and culture may also increase the difficulty of handling personality problems within a project. Team building is harder without frequent personal contact. Personality conflicts may be harder to resolve if they occur within inter-organisation relations. It may well be impossible to remove someone who is uncooperative from a project.
- Although the project manager normally reports to a steering committee or project board, which is formally responsible for the direction and success of the project, in practical terms, it falls to the project manager to ensure that the management of each of the partners is committed to the project throughout its lifetime. The project manager, therefore, in effect, has several, possibly conflicting, masters who need to be persuaded of the continuing value of what is being done.
- Intellectual, creative, work normally goes well if there is a critical mass of people working together. Ideas and understanding grow if they can be bounced around a group. A project that is split across several sites may not achieve this critical mass unless significant investment is put into travel, telephone calls, and electronic communication.

In addition, Herbsleb et al. (2005) introduced several project management and collaboration management issues that were in proportion with the previously described problems. For example, their lessons learned based on Siemens' experiences of nine projects pointed out that communicating what is desired of the service provider is likely to be more difficult, often much more difficult, than anticipated. They propose to develop ways of testing the service provider's understanding; especially, if schedules and plans seem optimistic, management should dig into details and make sure the service provider really understands what is required. From a project management viewpoint, they suggested to agree on the 'management infrastructure' of detailed milestones, tracking metrics, and reporting up front. Herbsleb et al. (2005) advised, '*Be prepared to be intimately involved in project management of staff at the service provider organisation. Either from the outset, or at the first sign of trouble, manage them as you manage your own staff, being aware of what they are doing on a day-to-day basis. Do not assume that the project management will be adequate until the service provider has a track record*'. Further, Herbsleb et al. (2005) emphasise the importance of communication and how communication shall be planned and managed in GSD. Utilisation of collaboration tools, travels, and shared development environments have to be carefully planned, implemented, and communicated amongst the partners of collaborative projects. This means that the project management process includes also many collaboration management tasks in global product-

development projects. For example, relations between collaboration partners, such as customer-supplier relations, need to be systematically planned and managed via appointed persons and activities in organisations. Moreover, responsibilities and authorities (e.g., decision-making authorities) should be made explicit between collaboration partners. That includes, for example, an organisation for coordinating the collaboration and for escalation of problems and decisions, involving management from each collaboration partner. In addition, a defined interface (person) from all collaboration partners is useful to have. Regular meetings between the partners to discuss situation are also useful.

2.3.3 Information needs for management

The success of software development depends highly on providing the right knowledge at the right time, at the right place, and for the right person (Ruhe 2003). Management of global product-development projects may require new or changed practices for project planning and tracking because of collaboration; for example, because of schedule dependencies that need to be managed. In addition, it is important to define clearly status reporting practices and change-management procedures, including reporting channels, decision authorities, and escalation channels. Based on MERLIN inventory interviews, identification of dependencies between partners (e.g., the interdependencies of the subsystem deliveries) and considering them in project schedules was seen as a critical issue. The dependencies should also be made explicit by defining responsibilities for delivery (who, what, when, to whom), authority to accept, as well as the acceptance procedure. The status of the dependencies should then be checked proactively.

Dullemond and van Gameren (2013), in their empirical study, identified the most important information needs during distributed software development. Table 4 summarises the list of the most important project-specific information items.

Table 4. Project-specific information items, based on (Dullemond and van Gameren 2013).

Most Important Project-specific Information Items	Importance
Technological agreements *e.g., on programming language, frameworks, and standards to use*	0.89
Requirements	0.94
Risks *(project specific)*	0.94
Process agreements *e.g., roles, stakeholders, and the process type*	0.88
Issues *(tasks)*	0.63
System under construction:	
Source *(repository)*	0.37
Build status e.g., build succeed /failed	0.63
Deployment status e.g., currently deployed version, is it running?	0.74
Planning:	
Deadlines	0.95
Meetings	0.84
Status:	
Hours worked on the project	0.42
Milestones	0.84
Phase of project *e.g., starting up, active, commissioning, or done*	0.58
Project-related communication with the customer *e.g., mail, phone calls, and transcripts*	0.95
Project-related communication with the team *e.g., mail, phone calls, and transcripts*	0.95

The table shows that quite a lot of information items were considered important or very important. While taking the importance indicator value of 0.9 as a cut-off point, the most important items were updates on requirements, risks, deadlines, customer communication, and team communication *from a project* where the engineer is was currently working. In addition, Dullemond and van Gameren (2013) found over 0.9 importance indicators from updates on information items of organisation-specific risks and personal contact information.

However, the information currently delivered by existing tools to project managers is not meeting their needs. In reality, managers must rely primarily on experience and intuition for critical decision making when data needs are not met, either because the tools are unavailable, too difficult to use, too hard to interpret, or they simply do not present useful or actionable information (Buse and Zimmermann 2010). In fact, decision making will likely become even more difficult, as software projects continue to grow in size and complexity. As a result, it has been pointed out that there is an urgent need for measurements of large-scale

software systems, a need that poses great challenge for computer science (Mingguang et al. 2009).

2.4 Measurements and metrics in GSD

Measurements and metrics create useful ways for controlling and managing a project during product development. Especially, in global product development measurements and metrics activities are emphasised. From the perspective of measurements and metrics interpretation, data collection from multiple sources even from different databases of various stakeholders is an essential process – the data collection should be highly accurate but as unobtrusive as possible. Actually, it is crucial to track and manage software development processes efficiently by continuously collecting and analysing measurement data. Thus, measurement automation is seen as a solution for continuously collecting and analysing the measurement data.

Management of a distributed product-development project has been proven more challenging than is the management of a single-site project. Based on the industry survey (Paper I), one of the most important topics for project management in distributed software development is detailed project planning and control during the project. In GSD this includes, for example, dividing work by sites into subprojects and establishing clearly defined responsibilities, dependencies, and timetables, along with regular meetings and status monitoring. That is why, in distributed projects, the systematic monitoring and reporting of the project work for providing support for decision making and project management is especially important, and measurement and metrics are important means to do that effectively.

Traditional measurement has been understood as an information gathering process, where measurement data consist of numeric data or a pre-classified set of categories. Because software metrics can consist of several measurement data items individually or in combination, measurements and metrics have been strongly linked to the development tools used during the production. Globally distributed development generates new challenges and difficulties concerning measurements. For example, the gathering of the measurements data can be problematic because of different development tools and their versions; work practices can vary by project stakeholders; or the reliability of the gathered data can vary because of cultural differences, especially, in subjective evaluations. Thus, there are good reasons to gather the measurement data from multiple sources (i.e., from different tools and databases). Otherwise, for a successful measurement process, all stakeholders have to transfer data items manually to a certain tool or database from their own tools and databases. This kind of manual process requires extra effort, which may create frustration and mistakes or errors in the gathered measurement data.

Measurements bring several benefits for an organisation. The benefits include better time-to-market estimation via improved project and product management, better control over product development costs owing to a more visible

development process, and higher sales and improved customer satisfaction because of higher product quality (Lawler and Kitchenham 2003). However, the fact is that organisations usually cannot allocate enough time or resources to do the measurements properly. Therefore, automation of the measurement process is a critical success factor in carrying out the measurement programmes in a cost-efficient and systematic way. Measurements automation allows organisations to perform sophisticated measurements and investigate complex phenomena, which would otherwise not be tackled because of the large amount of work required. Thus, measurements automation enables the use of more advanced measurement procedures (Batagelj et al. 2008).

2.4.1 Development-environment-related challenges

The business environment of collaborative and distributed development differs fundamentally from that of local development. Collaboration 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 involved can combine in any one of several different business relationships and for very different periods, ranging from the duration needed to exploit a particular innovation or business opportunity, to a much longer-term on-going relationship (adopted from (Welborn and Kasten 2003)). Even though GSD offers many opportunities for organisations, it is also highly challenging. General risks for any mode of collaboration influence the openness of communication between partners, trust between partners, agreement on the intellectual property rights, the reliability of the partners' development schedule, and the clarity of assignments or specifications of the work under contract (Hyysalo et al. 2006). For example, problem hiding may be a difficulty in customer-supplier relations.

From a measurement and metrics viewpoint, the challenges derived from business environments include a lack of a transparent and unified monitoring process across all the sites, of traceability between work items, and poor or no interoperability of the tools used. Organisations have their own processes, practices, and tools (tool versions), and they are unwilling to change them. For example, certain work practices or cultural differences themselves cause challenges while collecting or interpreting measurement data (Komi-Sirviö and Tihinen 2003). If no historical data is available, the utilisation of the data across the sites is difficult or even impossible. In addition, lack of communication and problems in knowledge management and transfer create challenges.

Now, several tool providers supply various solutions for managing collaborative projects. However, most of the solutions have been developed to communicate only within tools from the same provider. In GSD, partners and stakeholders usually change according to the new collaboration setting. Thus, there are several legacy tools used in the companies that they are not willing to change. For example, some of the development tools can even be tailored to individual partner

needs. Moreover, the cost of such an investment is sometimes higher than companies can afford. The main challenges that were found while managing distributed collaborative-software-development projects are introduced as follows (Eskeli and Maurolagoitia 2011):

- *Learning curve.* People do not like to change the technology or tool on which they are developing a product. They are not familiar with new tools and technology, and they tend to resist when they have to change their working platform. Thus, the learning curve will be very low.
- *Poor interoperability between tools.* For example, when data is moved from a requirements management tool to a test tool, defects are easily introduced. Different partners may use different tools and when the data is integrated, it can result in many errors and defects.
- *Responsibilities and roles are not properly defined*: People do not know to whom they should report or who is responsible for what task. In such cases, the resulting problem was that the escalation mechanisms were not clearly defined.
- *Lack of knowledge of standard solutions.* Sometimes developers start creating the same solution, that is, one that has already been implemented as a standard solution, thus leading to a total waste of resources. Before starting development, a proper background check for the product needs to be completed, as developers do not always know why or for what purpose they are creating a product.
- *Resource management*. It is very difficult to manage resources in a multisite-project environment. People with the right skills and real competence are always busy with lots of work. Therefore, it is very difficult to start work when proper resources are not available or the information about them is unavailable.
- *Cost of currently available tools.* Now, the market has a number of tool providers that supply solutions for managing collaborative projects. Most of these solutions will only communicate using the tools from the same provider, which limits the options organisations have. In collaboration, there can be several various tools and tool providers involved. Thus, the need for finding a common solution is obvious but it should require investments from some partners. Unfortunately, the costs of this kind of investments are typically higher than companies can afford.

During the PRISMA project, tools as well as needs for development tools and processes that require more support in GSD were investigated. In considering tools, the kind of artefacts, tasks, or items that would need integration were analysed or views between the partners during collaborative and distributed development, and what would be the most important issues that should be integrated in a proposal tool integration solution. The needs focused on

requirements capture and review processes, traceability needs, testing processes, project management, and controlling tasks (including metrics capturing and analysing needs). The used tools and the needs for support in GSD were as follows (Eskeli and Maurolagoitia 2011):

- *Requirements capture.* The tools used in this process were TRAC, SQS's AgileREQ, Focal Point, MS SharePoint, MS Excel sheets, and DOORS. The needs for a requirement capture tool include having a common and unique requirement repository that implements traceability mechanisms with other information items (e.g., other requirements, bug reports, related test cases, and requirement information sources).
- *Requirements review*. The tools used in the process were TRAC, some proprietary tools and MS SharePoint. A tool that helps during this vital activity is needed, as it offers the possibility to keep track of a full history of review comments and access to metrics (e.g., review effort or defects found).
- *Traceability*. The tools used were Subversion and TortoiseSVN, DOORS, SVC, SQS's Test WorkFlow, and Excel sheets. Because traceability was seen as a key task in collaborative software development, the main need of the partners was to have the possibility to assign and review the traceability amongst all of the information items in the project (e.g., requirements, test cases and reports, bug reports, and others).
- *Testing.* The main tool used in the process was SQS's Test WorkFlow. The needs for testing tools include a way to determine which tests are required to validate a release, specification of various test configurations, being able to repeat standard set of tests quickly, and the ability to create automatically test reports, easy definition of test scripts, continue regression if an unexpected problem or defect is encountered, and to have automatic defect reporting with attached information of test case, error condition, and test data.
- *Metric capture and analysis.* The tools used in this process included a custom self-developed tool, Excel sheets, and MS SharePoint. Tools to improve the reliability of the releases, and to decrease the validation times, collect automatically metrics and generate reporting graphs and include overview and detailed views are needed.

Further, the measurements and metrics challenges were also investigated in detail. The metrics used by PRISMA industry partners included, for example, test coverage and test results, effort spent, number of errors revealed, and requirement changes count. The metrics were related to the processes where tool support needs in collaborative development were highlighted. The tools used in measurement process were custom self-developed tools, Excel sheets, and MS SharePoint. In a collaborative distributed development, where development partners, methods, and tools vary by project, measurements automation and

transparency between the partners are understood to be vital and necessary (Parviainen 2012). In addition, the measurement process, from data definition and collection to data calculation and analysis, should be made to be as easy and effortless as possible.

2.4.2 Metrics in the GSD literature

In the literature, several papers and books that discuss metrics (in general and specific aspects) have existed for decades. However, little GSD literature has focused on metrics and measurements or even discussed the topic. Bourgault et al. (2002) reported, 'Clearly, research into distributed projects' performance metrics and measurement needs more attention from researchers and practitioners so that it can contribute to the development and diffusion of well-designed management information systems'. However, years later, da Silva et al. (2010) reported a similar conclusion based on analysis of distributed software development (DSD) literature published between 1999 and 2009. They state as one of their key finding that the 'vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of DSD project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature'.

The papers that have discussed metrics for GSE usually focus on some specific aspect; for example, Korhonen and Salo (2008) discussed quality metrics to support the defect management process in a multisite organisation. Simmons and Ma (2006) discussed a software engineering expert system (SEES) tool, where the software professional can gather metrics from case tool databases to reconstruct all activities in a software project, from project initiation to project termination. Misra (2009) presented a cognitive weight complexity metric (CWCM) for unit testing in a GSD environment. Lotlikar et al. (2008) proposed a framework for global project management and governance, including some metrics with the main aim to support work allocation to various sites. Peixoto et al. (2010) discussed effort estimation in global software engineering, and one of their conclusions is that 'GSD projects are using all kinds of estimation techniques and none of them is being consider as proper to be used in all cases that it has been used', meaning, that there is no established technique for globally distributed projects. In addition, some effort has also been invested in defining how to measure the success of GSD projects (Sengupta et al. 2006), and these metrics mainly focus on cost-related metrics, which are done after project completion.

2.4.3 Measurements and metrics related challenges

In the literature, many problems and challenges have been identified that might reduce or even eliminate concern over measurements. For example, not enough time is allocated for the measurement activities during a project or not enough

visible benefits are gained by the project doing the measurement work (e.g., data are useful only at the end of project, not during the project). In addition, the 'metric enthusiasts' may define too many metrics, making it too time consuming to collect and analyse the data. Thus, it is beneficial (Umarji and Shull 2009) to define core metrics to collect data across all projects, to provide benchmarking data for projects and to focus on measurements that come naturally out of the existing practices and tools. Further, metrics visualisation (e.g., through charts or graphs) is a powerful means for interpreting measurement data.

In GSD, the systematic controlling and status reporting of the project work is especially important, and measurement and metrics provide important means to do that effectively. However, in daily software engineering work, measurements and metrics appear unfamiliar, and their benefits are unclear (Umarji and Shull 2009). Even though now almost every project development artefact can be measured with a high degree of automation, efficiency, and granularity, software development continues to be risky and unpredictable. For that reason, Buse and Zimmermann (2010) have claimed that there must be disconnect between the information needed by project managers to make good decisions and the information currently delivered by the existing tools.

More than ten years ago, it was realised that distributed software-development projects are loaded with challenges, and measurement data is needed to back up decision making (Komi-Sirviö et al. 2001; Lawler and Kitchenham 2003). In addition, it was feared that there was a very high risk of getting no data at all, or of getting incomplete or inaccurate data, which would lead to a hazardous decision-making situation, where project decisions regarding execution could be based on feelings and assumptions of the status, or even worse, on false data (Lawler and Kitchenham 2003). Similarly, Lawler and Kitchenham (2003) pointed out that the measurement programs, at that time, suffered both from a lack of metrics standards that reduce data comparability and from invalid and missing data that causes delays in data analysis and reduces confidence in any reports' validity. He argued that companies could avoid those problems by fully automating their measurement programs and integrating them directly with their software-development support tools (Coman et al. 2009). Now, ten years later, the methods, processes, and tools for collecting and analysing measurement data have substantially improved, and large volumes of data and many types of metrics exist for project managers. However, Coman et al. (2009) claim that software projects are still difficult to predict and risky to conduct.

The problems are caused by the high demands that force designers to solve complex problem by building large-scale software. In addition, the increasing complexity and expanding scale leads to many difficulties and uncontrollable quality problems. From the developers' perspective, the lack of comprehension and effective evaluation makes it difficult to know their previous work and bring useful suggestions for future development (Mingguang et al. 2009; Zhang et al. 2008). For example, Korhonen and Salo (2008) stated that defects are a significant factor in software quality and in large-scale software systems where the number of defects can be very high. However, the defects are difficult to manage,

especially in a multisite organisation. Therefore, software measurement plays an important role in software engineering (Mingguang et al. 2009), and it is a powerful tool for managing software projects (Soubra et al. 2011), even if there are several challenges related to measurements.

The measurement data item consists of numeric data (e.g., efforts, schedules) or a pre-classified set of categories (e.g., severity of defects: minor, medium, major). Software metrics can consist of several measurement data items, both individually or in combination. Further, metric visualisation is a visual representation of the collected and processed information about software systems. Typically, software metrics are put into a visual presentation to ensure that the information is presented in a meaningful way that can be understood quickly. For example, visualising metrics through charts or graphs is usually easier to understand than long textual or numerical descriptions. In fact, the interpretation of measurements data is more complicated in GSD than it is in single-site projects. For example, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that comparing them is impossible. From an interpretation perspective, the visualisation of metrics is an essential task: including, for example, a selection of the most relevant metrics to be visualised, the visualised type, and the utilised effects of visualisation, such as trend lines or colours.

Now, there is so much data available in different databases about distributed projects that it is impossible to browse manually through it all. Thus, the data collection and metrics visualisation should be automated whenever possible. In addition, automated data collection increases the validity of data and decreases the overhead of the measurement program, which also decreases developers' resistance to measurement (Coman et al. 2009).

On the contrary, as data collection has become a common practice in the domain of software development engineering, the problem is not the availability of data, or the ability to access them, but rather the mining of the relevant data. Namely, most software development organisations have huge challenges to effectively discover the relevance of the data and ultimately, potential patterns, for example, when applying them to new releases or trying to enhance their products' quality. Hence, Altidor et al. (2009) suggested using data mining techniques to mine organisations' software repositories, by which they can build predictive models to ensure the quality of future software products or releases. Further, the software measurement allows engineering principles to be applied to software development, which provides an objective and quantitative base for process and technology decisions. Soubra et al. (2011) stated, 'For instance, software size gives a measure of the software product itself, and it can be used to obtain software development productivity ratios and to build objective estimation models for predicting project effort and duration. In practice, software size, measured in function points, for instance, is highly correlated with project work effort'.

Measurements are generally understood and recognised as a central and important factor for management. However, in daily project work in software production, measurements and metrics can be experienced as unfamiliar. Project

managers have argued that collect metrics for the organisation is time consuming, although they actually need to have metrics that are relevant to the project progressing (Umarji and Shull 2009). They have also suggested that there usually is not enough time budgeted for measurements, and that is why it is difficult to get approval from stakeholders for this kind of work. In addition, globally distributed development generates new challenges and difficulties related to measurements. For example, the gathering of the measurements data can be problematic because of the use of different development tools and their versions, because work practices with related concepts can vary by project stakeholders, or because the reliability of the gathered data can vary because of cultural differences, especially, in subjective evaluations. It has been also noted that distributed projects are unique in practice (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project), hence comparisons between the projects is impossible. Further, the interpretation of measurements data is more complicated in distributed projects than it is single-site projects. Special attention has to be focused on the training of the measurement data collection to ensure common understanding of them (e.g., used classifications). In addition, as measurements also tend to guide people's behaviour, it is important to ensure that all are aware of the purpose of the metrics (i.e., it is not to measure individual performance), specifically in projects distributed over different cultures.

2.5 Challenges in current measurements practices

Currently, collaborative software development is one of the most common approaches adopted by many software engineering practices. However, GSD generates new challenges and difficulties associated with the measurements. As introduced before, the gathering of the measurements data can be problematic when different development tools or their versions are used, work practices with related concepts can vary by project stakeholders, or the reliability of the gathered data can vary owing to cultural differences, especially, when doing subjective evaluations. There are also some obvious demands to gather measurement data from multiple sources.

In practice, measurements and metrics provide the means for managing dependencies and traceability items across sites during collaborative work. Because software metrics can consist of several measurement data items individually or in combination, measurements and metrics have been strongly linked to the development tools used during the production. Buse and Zimmermann (2010) pointed out that the information currently delivered by existing tools to project managers is not meeting their needs. Project managers must rely primarily on experience and intuition for critical decision making when data needs are not met, because either tools are unavailable, too difficult to use, too hard to interpret, or they simply do not present useful or actionable information. Measurements and metrics could create useful means for controlling and managing a project during

global software development; hence, those activities should be especially emphasised in GSD. However, the fact is that organisations usually cannot allocate enough time or resources to do the measurements properly. Therefore, automation of the measurement process is a critical success factor in carrying out the measurement programmes in a cost-efficient and systematic way.

In summary, some of the challenges relating to measurements and metrics are cumulated from measurements practices, metrics, and tools, whereas others are caused by people, and some result from collaboration settings. In addition, GSD settings affect the current measurement practices by producing new challenges or complicating old ones. For example, cultural differences and their consequences as introduced by Hofstede (1984) or Lewis (1999) need to take consideration while communicating measurements and metrics within project members and stakeholders.

In the following list, challenges encountered in current measurement practices have been summarised and described in detail, from the GSD viewpoint:

- ***Challenge 1 (C1): Relevance of measurements in relation to project progress***. Generally, partners and stakeholders change according to new collaboration setting. All partners or stakeholders have their own needs for measurements. Moreover, there is a lack of a transparent and unified monitoring process across all of the sites and traceability between work items. Thus, in GSD, the relevance and purpose of measurements in relation to project progress are more difficult to understand than they would be in a one-site project.

- ***Challenge 2 (C2): Extra work budgeting – Metrics design/selection & Data collection.*** In GSD, there are needs to gather measurement data from multiple sources (i.e., from partners' different tools and databases). Without automation, subcontractors are required to collect measurement data and manually transfer data items to a certain tool. Without extra budget, partners cannot allocate enough time or resources to do the measurements properly. In addition, this kind of manual process creates frustration and mistakes or errors in the data. However, in GSD, demands for dynamic measurements are substantial as the metrics need to be defined case by case and they should be followed on a daily basis.

- ***Challenge 3 (C3): Data reliability issues caused by tools.*** In GSD, there are several legacy tools used in the companies, and companies are not willing to change them. Some of the tools are even tailored to individual partner needs. This means that there is no or poor interoperability of used tools. In addition, the used tools vary based on collaborative setting. Thus, in the worst cases, the measurement data are manually collected by stakeholders, which leads to extra work (C2) as well as poor understanding of measurements relevance (C1).

- ***Challenge 4 (C4): Data reliability caused by human beings.*** In GSD, work practices can vary by project stakeholders, and the congruity and

even the reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. The challenge is the same as that faced in one-site development, but it needs to be taken continuously into consideration, as partners and stakeholders change according to the new collaborative setting.

- ***Challenge 5 (C5): Extra work budgeting – metrics interpretation.*** Interpretation of measurements data generally requires extra work. In GSD, the interpretation and decision making based on the measurements results are more complicated than they are in one-site development. For example, the decision making requires interpreted information gathered from several sources (/metrics). Thus, the interpretation should be made as easy as possible, i.e., using combined metrics and visualised graphs.
- ***Challenge 6 (C6): Training needs.*** In GSD, cultural differences as well as stakeholders' views for measurements shall be considered in more detail while planning and implementing trainings. With successful trainings, it is possible to reduce data reliability problems caused by human being.
- ***Challenge 7 (C7): Measurements affect to behaviour.*** This challenge is same in one-site and in collaborative settings: the measurements should not affect to people's behaviour. This requires careful metrics design but also trust between partners. Accordingly, just openness of communications between partners and trust are not axiomatic in GSD.
- ***Challenge 8 (C8): Metrics ethics.*** Measurements should be focused on objectives, not a certain person: do not measure individuals' performances. The challenge is same in one-site and in collaborative settings.
- ***Challenge 9 (C9): Responsibilities and roles are unclear.*** Because of several partners with their diverse work practices in GSD, there can be unclear assignments or specifications of work. In the worst cases, unclear responsibilities and roles can cause a situation where people do not know to whom they should report or who is responsible for each task. In fact, unclear responsibilities often lead to the situation where no one takes up the task.
- ***Challenge 10 (C10): Continuous changes.*** In GSD, partners and stakeholders change according to new collaborative setting and that itself provides new challenges for measurements and metrics: dynamic, reliable, and up to date support for the decision-making process is needed.

While studying the GSD related challenges introduced above in more detail, it can be noticed that some of the challenges are strongly dependent from each other. In fact, the challenges describe the situation from different viewpoints; those perspectives are important to take consideration while trying to find solution to measurements problems in GSD projects.

In GSD, partners and stakeholders change often according to the new project and collaborative setting. Hence, no stabile single solution is appropriate for all situations in GSD. Instead, while examining metrics and measurements in GSD, the wholeness must be considered, which means that challenges in current measurement practices need to be analysed from the viewpoint of managing GSD projects. Thus, the research was focused on identifying the most useful metrics set for providing appropriate information to respond to knowledge-management and communication needs as well as project management needs in relation to the tools used in GSD projects. Thus, the challenges caused by the dynamic environments of GSD projects influence the measurements and metrics. Hence, the need for measurements that are more dynamic becomes obvious.

3. Measurement-based management of GSD projects

In traditional measurements programs metrics are defined at the beginning of the project and then measurement data has been collected in pre-scheduled periods, e.g., bi-weekly or once a month. In GSD projects, measurement process and metrics have to be more dynamics because of development processes are rapidly changing due to various stakeholders and tools involved the collaboration. In this thesis, dynamic measurements are defined as measurement actions where metrics are defined or updated based on needs of each project and demands of each project's collaboration settings. In addition, metrics data is collected and analysed continuously from various tools and databases, even from stakeholders' databases, and measurement data is analysed and visualised for easy to read format. In dynamic measurements, metrics, used databases, and analysed results of measurement data can be created, updated and changed dynamically based on managerial needs of each project, the different phases and tasks, as well as the management needs of various stakeholders in collaborative work.

Dynamic measurements create a base for measurement-based management of GSD projects. Generally, measurements are seen as a supportive role in software engineering. In GSD, projects' controlling and managing activities are more and more important. Distributed and collaborative product development forces to pay attention to management and controlling activities for achieving awareness and knowledge of distributed activities within the project and even over the projects in an organisation.

In this section, the construct of measurement-based management of GSD projects are discussed by describing demands and requirements for dynamic measurements in Subsection 3.1 and by introducing an experimental implementation of dynamic measurements in GSD setting with industrial experiences in Subsection 3.2. Finally, the benefits of dynamic measurements are summarised in Subsection 3.3.

3.1 Dynamic measurements in GSD projects

Typically, organisations may have a set of metrics that are followed from all projects. In addition, those metrics may lightly be updated or even somehow expanded based on demands of development projects. However, in GSD projects, the demands for dynamic measurements are substantial and more often expressed: the metrics need to be defined case by case, and they should be followed even daily based, for example. In addition, it has been proved that lack of communication cause challenges to information and knowledge management and transfer in collaboration environments. In practice, the management of GSD projects require well-optimised automated and real-time indicators that provide up-to-date visibility to the stakeholders' progresses and results. Thus, the dynamic measurements are a key role while increasing transparency and information sharing during collaborative distributed software development. For achieving most optimal and well-balanced wholeness of the dynamic measurements, knowledge related factors relation to the team activities need to be carefully examined.

3.1.1 Knowledge management and transfer in GSD

Herbsleb et al. (2005) and MERLIN industrial inventory (Hyysalo et al. 2006) point out the importance of communication in GSD: communication shall be planned and managed, e.g., by utilising of collaboration tools, travels, and shared development environments. Those activities need to be carefully planned, implemented, and communicated in within the partners of collaborative projects. Because lack of communication and problems in knowledge management and transfer create many challenges in GSD, the examination of challenges from a cognitive perspective was performed. During the MERLIN and PRISMA projects, from 2004 to 2010, in total 54 industrial case studies were studied and analysed. The aim was to establish solutions that take into account the knowledge needs from the viewpoint of different stakeholders in GSD. The analysis was performed by utilising the model of Noble (2004), which illustrates the relationship between knowledge (individual and shared understandings) and key team activities (team set-up and adjustment, group problem-solving, and synchronize and act). The team set-up and adjustment process includes analysing the mission, determining the required members and resources, and assigning tasks and resources. Some of this knowledge can be written down, but a large amount will remain 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, for example, brainstorm, evaluate, prioritise, identify solutions, or make decisions. To synchronise and act, team members draw on their knowledge to coordinate and help each other. All teams perform all of the team activities, generally moving from left to right, but they also switch back and forth among the activities depending on their immediate needs. Figure 4 illustrates the results of the analysis.

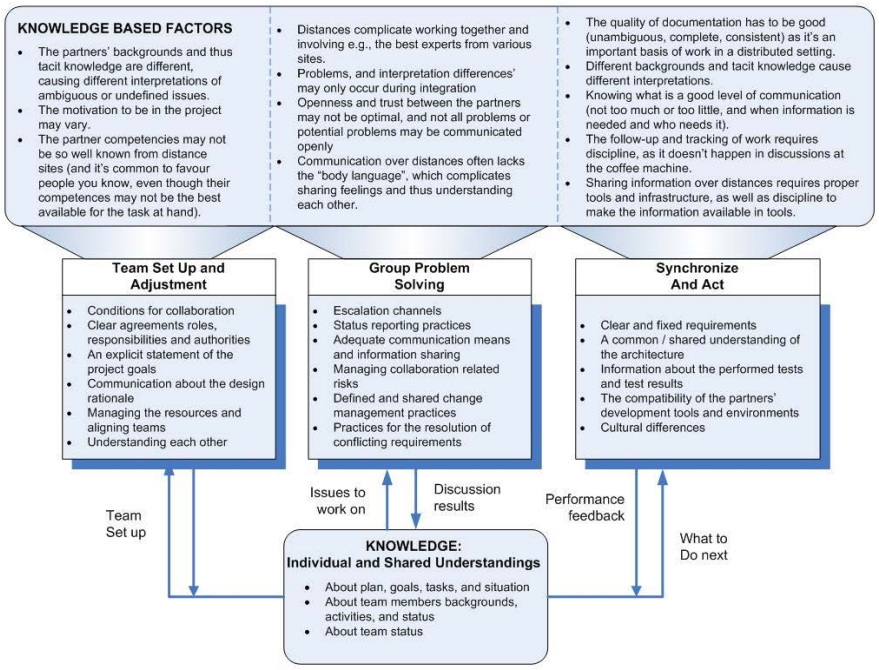

Figure 4. Knowledge related factors in relation to the team activities and solutions (Paper IV).

The upper part of the figure summarises the main knowledge-based factors for each key team activity proposed by Noble (2004). Then, in the middle part of the figure, examples of solutions are given according to the identified challenges. The knowledge-based factors and the related solutions are introduced in detail in Paper IV. The model of Noble (2004) was used as a framework for identifying and analysing the knowledge needs of distributed teams and stakeholders. This approach increases the visibility of knowledge based requirements and challenges, thus it makes possible to take them into account while carrying out improvement actions, and utilising general knowledge management solutions more extensively to solve GSD challenges. The research results emphasised the knowledge needs of distributed teams and stakeholders by analysing the challenges encountered in GSD from the knowledge engineering viewpoint. Based on the analysis of the research (Paper IV) it can be argue that knowledge engineering holds a central role in order to succeed with globally distributed product development. The analysis showed that by understanding the nature and demands of the GSD, as well as the knowledge engineering challenges in depth, software organisations would be able to reduce the risk of failure and to make their operations successful.

As introduced in Paper IV, successful distributed software development requires both structured and disciplined software engineering and knowledge

management solutions. Communication management and the utilisation of effective substitutes for face-to-face communication have an important role in GSD, to ensure knowledge sharing. A careful execution of project planning activities, the exact definition and agreement of common rules, responsibilities, and tools used, can greatly contribute to a successful implementation. This means that project manager has to have a large amount of abilities and knowledge in addition to technical competence and skills, such as cultural knowledge and communication skills and particularly good project management capabilities. In addition, ensuring the availability of information during the project to all of the parties is essential for a successful project.

In GSD, a project manager may be far away from the development teams, 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. That is why, in GSD projects the systematic controlling and status reporting of the project work is especially important, and in practise, measurement and metrics provide important means to do that effectively. The research (Paper IV) emphasises the need of real-time and accurate information in managing GSD projects. In addition, Paper IV points out that metrics and measurements should increase transparency between teams and stakeholders involved the project. Further, Paper IV highlighted that knowledge sharing and lessons learned have to be taken consideration in order to analysis and interpret metrics and then to make correct conclusions from the measurements data.

3.1.2 Requirements for dynamic measurements

While identifying requirements and prerequisites for dynamic measurements in GSD, it is valuable to analyse GSD related challenges summarised before in Subsection 2.4.5. Table 5 introduces requirements for dynamic measurements derived from GSD-related challenges in current measurement practices.

Table 5. Requirements for dynamic measurements derived from GSD-related challenges in current measurement practices.

Descriptions of requirements for dynamic measurements
C1: Relevance of measurements in relation to project progress Since each stakeholder has their own needs for measurements and for controlling the project progress, it is important that metrics are planned and defined based on needs of each partner. The actual measurement data can be same instead produced indicators can change: the metrics contain the relevance information from each own viewpoint. In addition, the metrics need to be updated or even changed, e.g., if some other metrics indicate results that should be clarify or completed. Thus, the metrics set is good to keep as compact as possible. In fact, the smaller metric set is more effortless to acquire, and to update if needed. Thus, the relevance of measurements in relation to project progress is easier to ensure in GSD even if partners and stakeholders change according to new collaboration setting.

C2: Extra work budgeting – metrics design/selection & data collection

In GSD, the metrics need to be defined case by case and they should be followed daily based. This means that metrics set cannot be too large; instead, a compact metrics set is easier to adopt and update when needed. Because of needs to gather data from multiple sources, even from partners, different tools, and databases, it sets requirements for automating the measurements data gathering and the results visualising. Even if extra work for measurements must always be budgeted for, the automation can reduce the needed amount.

C3: Data reliability caused by tools

This challenge need to be investigated carefully since there are several legacy tools used in the companies, and they are not willing to change them in GSD. This is also a requirement for dynamic measurements: companies legacy tools have to be allowed. In addition, during the automation process of measurement data gathering and results visualising, the validity of data has to be ensured. In dynamic measurements, the visualised indicators themselves offer also means for checking correctness of gathered data (deviations are visible in graphs).

However, it has to be always ensured that agreed tools and data classifications are used in uniform way by all stakeholders in GSD. This is done via kick-off meetings, project meetings and specific communication and training sessions during the work.

C4: Data reliability caused by human beings

As described before (C3), in dynamic measurements, the visualised indicators themselves offer means for checking the correctness of gathered data. However, it has to be always ensured that the agreed-upon tools and data classifications are used in a uniform way by all stakeholders. Continuous and active communications are extremely important while ensuring data reliability issues caused by human beings. The reasons for measurements and metrics have to be clarified and communicated appropriately; for example, there are not aimed to measure performance of individuals, instead to get information for building the conception of wholeness. Only this way, it is possible to re-scheduled or re-allocate resources to difficult tasks in the right time.

C5: Extra work budgeting – metrics interpretation

In GSD, the interpretation should be made as easy as possible (i.e., using combined metrics and visualised graphs). This requires automation of measurement data gathering and results visualising processes as a whole or at least partially. From the metrics interpretation view of point, it is good to combine various metrics in the same graph and add limiting values with alarms while appropriate. Because of each stakeholder has own needs for controlling the project progress, it requires that the metrics combined to the results graph can be varied based on stakeholders.

C6: Training needs

In addition to measurements and metrics activities, the GSD settings require more-specific training activities focusing on cultural differences, communication channels, and tools, clarifying responsibilities, roles, etc. The automation of measurements can reduce some training needs but training and communication are still essential for a successful project. The dynamic measurements do not eliminate training needs in GSD.

C7: Measurements affect to behaviour

The challenge is same in one-site and in collaborative settings: the measurements should not affect to people's behaviour. Continuous and active communications is extremely important: the reasons for measurements and metrics have to be clarified and communicated appropriately. This challenge requires careful metrics design but also trust between partners. The dynamic measurements cannot influence on this challenge.

C8: Metrics ethics The challenge is same in one-site and in collaborative settings; the measurements should not measure individuals' performances. The reasons for measurements and metrics have to be clarified and communicated appropriately. This challenge requires careful metrics design but it is true that the dynamic measurements cannot influence on this challenge.
C9: Responsibilities and roles are unclear GSD settings require more-specific planning, training and communication activities relating to responsibilities and roles during the project. In dynamic measurements, data needs to be gathered from multiple sources even if partners' databases or tools, so measurements automation is required. Manual data collection in this kind of challenging environment is highly difficult to operate successfully. In practice, unclear responsibilities often lead to the situation where no one takes up the task, so the dynamic measurements can even bring out unclear assignments in GSD, if the measurements have been automatised.
C10: Continuous changes In GSD, partners and stakeholders change according to new collaborative setting and each stakeholder has their own needs for measurements and controlling the project progress. Thus, it is important that metrics are planned and defined based on needs of each partner. In addition, the metrics need to be updated or even changed, e.g., if some other metrics indicate results that should be clarify or completed. Thus, the metrics set is good to keep as compact as possible.

Based on descriptions of requirements for dynamic measurements (Table 5), it can be easily deduced that automation is one important solution to the requirements. However, it can be also concluded that the relevance of measurements from the viewpoints of various stakeholders is emphasised. The relevance of measurements effect on entities what to be measured and when. This requirement produces also new requirements for measurements and metrics, such as updating/change needs, visualised needs, interpreting needs. It can be summarised that real-time and visualised indicators are needed for the decision support, especially, when decisions have to be made in complex, uncertain, and dynamic environments. However, when trying any solution to meet GSD-related challenges in current measurement practices, there are several knowledge-management and knowledge-intensive perspectives that must be taken into consideration. For example, potential solutions should boost communication and knowledge transfer and facilitate communication between the teams, sites, and stakeholders involved the GSD project.

3.2 Implementation of dynamic measurements in GSD projects

During the PRISMA project, a tool integration solution, PRISMA Workbench (PSW), was developed. The tool integration solution is a proof of concept of the technical implementation that enables dynamic measurements and metrics in GSD. In this section, the produced solution is introduced with a reference scenario for describing the typical setting of collaborative work in GSD. The technical

implementation was necessary for gathering experiences of dynamic measurements as well as for proving eligibility of the proposed concept of dynamic measurements.

3.2.1 Technical viewpoints

The implemented PRISMA Workbench (PSW) was designed to support collaborative and distributed software development. The actual tool set selected for the tool integration solution consists of tools based on the principles of technical implementation introduced in (Eskeli and Maurolagoitia 2011). The PSW solution allows the connection of software development tools to create company specific software development environment instances. The proposed solution fills the gap that exists in current collaborative software development environments: it allows distributed teams to integrate their own existing tools and link data amongst them.

Eskeli and Maurolagoitia (2011) point out that the studied challenges related to the methodologies and tools during GSD projects are varied, complicated, and multi-dimensional. However, one of the major design goals of the PSW was to create a flexible and extendible solution that allows a configurable set of development tools to be tailored to the needs of individual partners or projects. The solution was designed and developed so that the legacy tools in the companies already in use could be easily integrated into the proposed tool integration solution. The benefit of this type of tool integration approach is that partners can continue to use the tools with which they are familiar. In GSD, continued use of familiar legacy tools was identified as an important factor for effective way of working that could be easily disturbed by a sudden change of tools.

Another primary goal for PSW was that the integration of new tools be made as easy to use as possible. To reach this goal the following steps were taken: implement integration mechanism for simple integration, provide example integrations, and created integration instructions. In practice, the connections between data were managed in PSW via traceability relations. PSW provides various real time views into data, e.g., the views create visibility into project's progress by gathering and formatting data from tools, managing connection between data, and concentrating this information into easy to read dashboards (Eskeli and Maurolagoitia 2011). Thus, PSW removes the need to create point-to-point integrations between each of the tools because the solution can act as a hub where tools are connected via the integration interface.

Communication related challenges play a central role in GSD. To support and facilitate collaborative, distributed development, PSW provides a means for asynchronous (e.g., chat) and synchronous (e.g., voice and video) communication by utilising features on an open source tool called OpenMeetings. The notification system provides users up-to-date information for any important events, like 'build status change', in the project. The example tools that were selected for the integration are presented in the following table (Table 6).

Table 6. Tools used in PRISMA Workbench (PSW).

Tool	Tool type	Role in the software development life cycle
TRAC[1]	Project management	Requirement and change management
AgileReq (proprietary tool)	Requirements management	Requirements management
Eclipse[2]	Integrated Development Environment	Software development (code editor, subversion client, build tool, etc.)
Subversion[3]	Configuration management	Software version control
Jenkins[4]	Continuous Integration (CI) Server	Continuous build & integration Unit test driver
CruiseControl[5]	CI Server	Continuous build & integration Unit test driver
JUnit[6]	Java unit testing	Unit testing
Open Meetings[7]	Communication tool	Communication (chat, voice over Internet, document sharing, etc.)
TestLink[8]	Test case management	Acceptance testing
Liferay[9]	Web portal software	User interface for the tool integration (traceability information, events, metrics, etc.)

Although PSW can be connected to several commercial tools or proprietary ones, the development team of the PRISMA project made a selection of open source solutions that provide support of the most typical software-development-process activities. By using these free tools, companies will be able to start working together even if they currently have no tools available for requirements-specification, development or testing tasks. However, some proprietary tool was also selected. For example, AgileReq tool was integrated based on the company's needs of the PRISMA project. In fact, one principle for the development work had been that in GSD, companies have their own practices and tools, and the tool integration solution should not enforce a specific process or tool set.

The tool integration solution was a server application developed in Java for building customised tool integration instances. The selected architecture was Service Oriented Architecture (SOA), which provided several services implemented by tools (e.g., project management, test management, version management, and

[1] http://trac.edgewall.org/ Accessed 15.10.2014
[2] http://www.eclipse.org/ Accessed 15.10.2014
[3] http://subversion.apache.org/ Accessed 15.10.2014
[4] http://jenkins-ci.org/ Accessed 15.10.2014
[5] http://cruisecontrol.sourceforge.net/ Accessed 15.10.2014
[6] http://junit.org/ Accessed 15.10.2014
[7] http://code.google.com/p/openmeetings/ Accessed 15.10.2014
[8] http://testlink.org/ Accessed 15.10.2014
[9] http://www.liferay.com/ Accessed 15.10.2014

requirements management). In addition, a set of core services like authentication, security, and management of traceability between work items were implemented in the server application. The architecture of the framework is shown in Figure 5. The tool integration server was implemented using Apache Tuscany[10], as it provided support for implementing SOA. In addition, it provided the required infrastructure for easy development and running of applications using a service-oriented approach. The user interface via a web portal was implemented with the open source enterprise portal software, Liferay, where all functionality was provided via portlets.

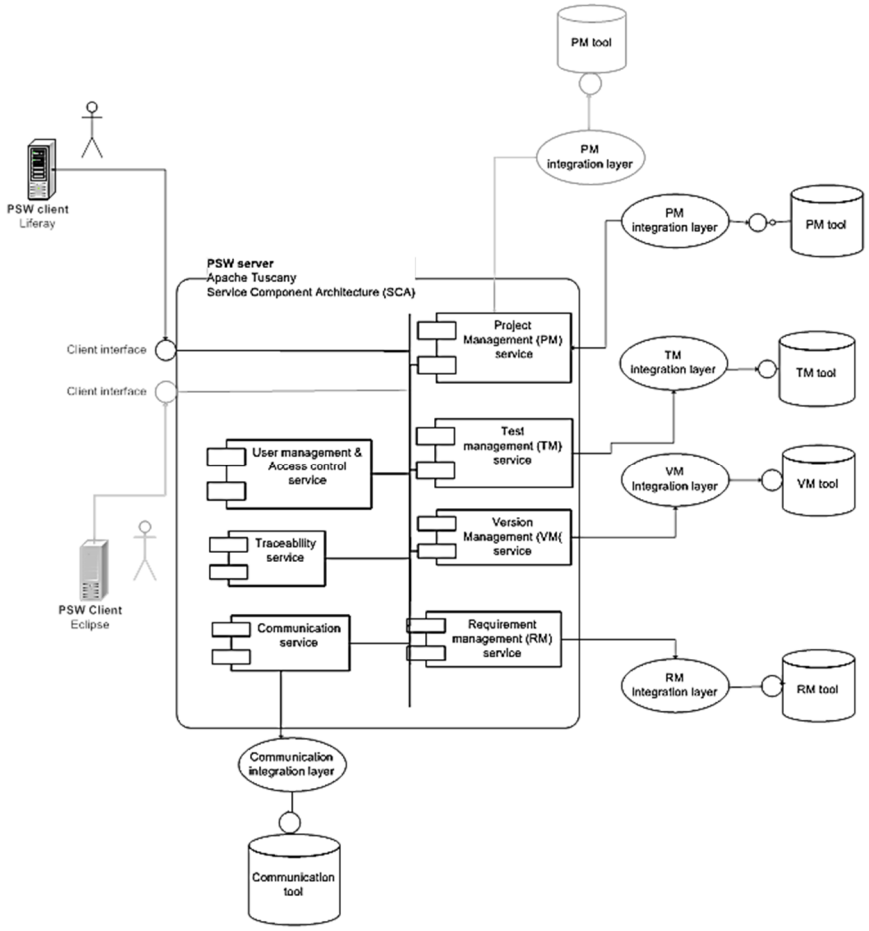

Figure 5. The architecture of the implemented tool integration (PSW).

[10] http://tuscany.apache.org/ Accessed 15.10.2014

In the tool integration, the connections between work products (i.e. requirements) can be managed via traceability relations. The relations indicate some type of dependency between work products. For example, a relation between a requirement and a test case can indicate that the given requirement is validated by the related test cases. The PSW provides the means to identify these relations and the views to visualise them. The relations can also be used in reports, e.g. to show requirements test coverage, requirements test status, etc. This kind of feature is especially useful in maintaining control of the project when the work products that should be logically dependant are managed in separate tools (and/or sites). The PSW provides the visibility of how the project is running and what every group is doing to the whole development team almost as if everybody would be working in the same room.

In the example scenario, Eclipse is used to develop source code for Android application based on the requirement tickets stored in TRAC. Subversion version-management tool is used to commit the source code changes to the repository (recorded as new change sets). Information about change sets is kept in TRAC for each requirement ticket. After the source code change, the Jenkins CI server creates a new 'build' and runs unit tests for it using JUnit. If the build is successful, the acceptance tests are run (at a different site) and the results are stored in the TestLink. All of these tools except Eclipse are integrated via the integration framework, and their status can be monitored from the project portal (Liferay).

3.2.2 Measurements viewpoints

The tool integration solution, PSW, was used as a proof of concept of technical implementation that enables dynamic measurements and metrics in GSD. The PSW enables automated measurement data gathering and transferring between various software-development tools in collaborative settings. This makes information sharing easier between partners without having to change significantly the current environment, tools, and processes. In addition, there was also a metric set defined and implemented in the PSW. The metric set was designed with tight co-operation with industrial partners involved the PRISMA project. In fact, metrics were selected based on successful experiences of their use in real GSD projects (Paper IV). This metrics set was especially aimed to provide means to respond proactively and in real-time to potential issues in the project. From viewpoint of measurements, it means that several metrics were combined and measurements results were visualised with appropriate graphs. In fact, the PSW solution provides several real-time views, visualised metrics, into data that had been stored into various data sources even in separate stakeholders' databases. The metrics were designed to increase transparency by providing visibility to the issues encountered in GSD.

The data visualisation need was one driver for the views developed to the PSW. The following figure (Figure 6) shows with a metric example, *budget status*,

how the tool integration solution can integrate data from various tools and provide a visualised view of the status.

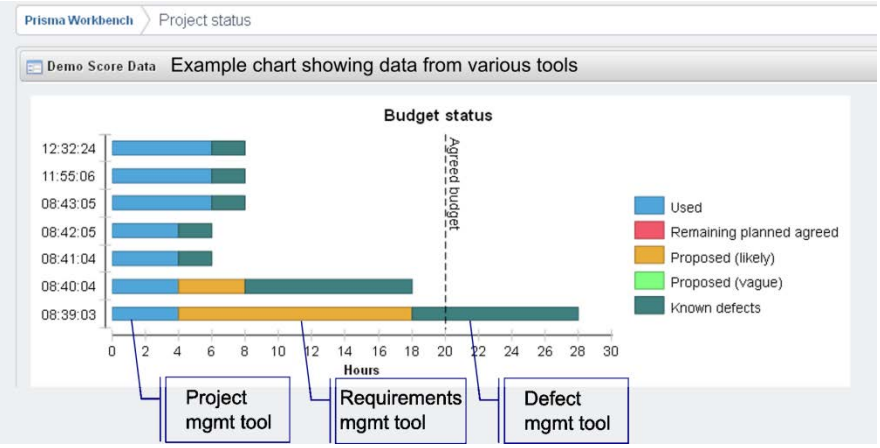

Figure 6. A graph generated by PSW.

From viewpoint of dynamic measurements in GSD, the budget status metrics (shown in the Figure 6) demonstrates the indicator, where measurement data are collected from various tools such as:

- project management tools (used effort),
- requirements management tools (effort estimated for proposed vague, proposed likely and remaining planned & agreed requirements), and
- defect management tools (effort estimated for known defects).

The tool integration solution provides the means to identify these relations and the views to visualise them. In GSD, these kinds of attributes are typically processed and managed in separate tools and even in different sites. Thus, in the worst cases, these attributes are manually gathered and reported for measurements purposes. Instead, the tool integration solution increases transparency between partners and teams by providing visibility from the actions of each development group.

In GSD, the example metric offers better visibility to the stakeholders' progresses and results by providing real-time and visualised information. The budget status graph shows actual costs of the project in portion with the agreed budget over a time. The metric also gives several dynamic indicators of estimated prospective costs in each pre-scheduled period. The bars summarise amount of costs, and each bar is composed from different cost-related data. The first block (blue) describes actual cumulative costs of the project. The agreed budget for the project is shown clearly as a black dash line in the middle of the graph. The red block would describe remaining planned cost based on effort estimated for requirements that have been accepted for implementation but not implemented.

The orange block indicates proposed cost that can be seen very likely costs for the project. These costs are based on effort estimated for the proposed requirements that are thought likely to be implemented, for example, a customer will want them. The green block would describe proposed but vague costs for the project. These costs are based on effort estimated for the proposed requirements, which the likeliness for implementation is not known. Instead, the dark green block indicates potential costs for the project, so-called 'Known defects' costs. In GSD, the example real-time metric provides great advantages for daily project management and decision making because the views are generated automatically and continuously, for example, by daily based. Thus, the project manager can control several processes – used efforts, requirements, and testing processes – via one visualised graph even beyond partners' borders.

The developed version of the tool integration solution contains a set of metrics that were identified as being beneficial for stakeholders during GSD projects, introduced more detail in Paper V. In addition, partners were able to define new views by utilising measurements data gathered via the tools integrated to the PSW. This was one important requirement for the solution because changes are typical in GSD projects and so new or changed circumstances may require new and updated indicators. Thus, PSW provide a proof of concept of measurement-based management in a collaborative setting.

The real benefits of the proposed tool integration solution and generated views by PSW can be achieved when several views can be read simultaneously. This is also an important factor for measurement-based management of GSD projects. Controlling and managing of distributed collaborative project can be supported with real-time dashboard views of project activities and progress. Following figure (Figure 7) illustrates how metrics set can be shown as a dashboard view providing visibility into project's progress. The figure has been generated by the implemented PSW solution. From confidential reasons, the shown indicators in Figure 7 have been generated from a demonstration project, not a real industrial project.

Figure 7. A dashboard illustration of the project status metrics.

In general, the interpretation of a project's comprehensive status needs a variety of metrics information – such as requirements status, progress status, testing status, and budget status – for making decisions based on the data. In addition, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires 'super-balancing' – how to come to the right corrective action if for instance, on the one side, the percentage of not accepted requirements is high, and on the other side, the number of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, taking into account the subjectivity of metrics, such as effort estimation, differences between backgrounds of the people (cultural or work experience) in different sites may affect the result.

In GSD, it is important to note that automated capturing reduces the chance of variations caused by differences in recording the metrics data in different sites. Additionally, some metrics correlate with each other, for example, metrics relating to tests correlate with metrics about requirements, and that needs to be taken into consideration while analysing measurement data.

When implementing the measurement framework (such as the tool integration solution) for dynamic measurements there are several issues to be taken consideration like defining and selecting metric set, identifying the optional metrics combinations, automating measurements (partially or completely), and visualising and interpreting the measurements results.

Communication and knowledge play an important role in GSD project and this aspect need to be carefully considered and planned in the technical implementation. In practice, project manager needs both technical competence and skills and managerial abilities that can be supported by dynamic measurements, but she/he needs also knowledge and communication skills. For example, taking into account the subjectivity of metrics, such as effort estimation, differences between backgrounds of the people (cultural or work experience) in different sites may affect the result. The fact is that dynamic measurements can make visible some problems or deviations and this way increase transparency between partners. In reality, the metrics can also indicate some inconsistent results, so in those cases it requires trying to find real reasons by interpreting other metrics related to the activities.

The results of using the PSW pointed out that the tool integration solution – measurement-based management framework – can really be utilised while managing and controlling GSD projects. Anyway, project management includes knowledge management activities that need to be taken consideration, especially, in GSD where partners, contact persons, and tools varied a lot. Tools or automation cannot compensate face-to-face communication or offer knowledge that needed in different collaboration settings. Instead, tools and automation can be an enabling role or make communication easier in GSD projects. In practice, the measurement-based management can offer views to the progress of partners' project or tasks and thus it can make visible possible problems or challenges in development tasks, for example. The measurement-based management with dynamic measurements creates transparency between the partners and teams in GSD projects.

3.2.3 Examples of industrial cases

As introduced before the metrics have to be defined case by case and they should be followed daily based in GSD. Thus, in practice, there are needs for examples of metrics as well as industrial experiences of their use (~the best practices). Therefore, during the PRISMA project industrial case studies were performed for identifying potential metrics set for GSD projects. The aim was to identify a basic metric set that can be enhanced, update or modified based on the needs of the certain GSD projects. Paper V points out that the successful metrics shall be easy to capture, they shall be able to capture from used tools 'for free' and they shall be

quickly calculated at regular intervals. Thus, the metrics set is good to keep as compact as possible since the smaller metric set is more effortless to be adopted as well as combinations of metrics are easier to be updated. Then also, the relevance of measurements in relation to project progress and stakeholders' needs is easier to ensure even if partners and stakeholders change according to new collaboration setting. This requires automation of measurement data gathering and results visualising processes as a whole or at least partially.

The metrics set that was successfully used in distributed product development was studied during the PRISMA project. The produced metric set is introduced more details in Paper V. The main purpose of the study was to offer a set of essential metrics with experiences of their use in GSD. Thus, the amount of the metrics was knowingly kept as limited as possible. The proposed metric set was generated based on experiences and needs of PRISMA project industrial partners (Eskeli and Maurolagoitia 2011) as well as studies of information needs in GSD (Dullemond and van Gameren 2013) and knowledge of the most critical project management activities for successful projects reported in (Jones 2004). The proposed metrics with their notations and definitions are summarised in the following table (Table 7):

Table 7. Metrics set successfully used in GSD, based on Paper V.

Metric	Notation	Definition
Schedule: -Planned -Actual	 $D_{PLANNED}$ D_{ACTUAL}	The planned/actual date of delivery (usually the completion of an iteration, a release or a phase)
Personnel: -Planned -Actual	 $\#FT_{PLANNED}$ $\#FT_{ACTUAL}$	The planned/actual number of full-time persons in the project at any given time
Documents: -Planned -Proposed -Accepted	 $\#Docs_{PLANNED}$ $\#Docs_{PROPOSED}$ $\#Docs_{ACCEPTED}$	The number (#) of planned /proposed/accepted documents to be reviewed during the project
Requirements: -Proposed -Accepted -Not implemented -Started -Completed	 $\#Reqs_{PROP.}$ $\#Reqs_{ACCEP.}$ $\#Reqs_{NOT\_IMPL}$ $\#Reqs_{STARTED}$ $\#Reqs_{COMPLETED}$	The number (#) of - proposed requirements - reqs accepted by customer - not implemented reqs - reqs started to implement - reqs completed
Change Requests: -New CR -Accepted -Implemented	 $\#CRs_{NEW}$ $\#CRs_{ACCEPTED}$ $\#CRs_{IMPL.}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented

Tests: -Planned -Passed -Failed -Not tested	 $\#Tests_{PLANNED}$ $\#Tests_{PASSED}$ $\#Tests_{FAILED}$ $\#Tests_{NOT\ TESTED}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
Defects -by Priority: e.g., Showstopper, Medium, Low	$\#Dfs_{PRIORITY}$	The number (#) of - defects by priority during the time period
Effort: -Planned Effort -Actual Effort	 $E_{PLANNED}$ E_{ACTUAL}	The planned/actual effort required of any given iteration of the project
Size: -Planned size -Actual size	 $SIZE_{PLANNED}$ $SIZE_{ACTUAL}$	The planned/actual size of each iteration can be measured as SLOC (source lines of code), points, or any other commonly accepted way
Cost: -Budgeted -Expenditure	 $COST_{BUDGET}$ $COST_{ACTUAL}$	The budgeted cost/actual expenditure for any given iteration
Velosity: - Planned / Actual story points	 $\#SP_{PLAN}$ $\#SP_{ACT}$	How many story points (SP) are planned to be /actually implemented of any given iteration of the project?
Productivity:	$\frac{\#SP_{ACTUAL}}{E_{ACTUAL}}$	Number of acutally implemented story points per used effort for each sprint /iteration

The metrics related to schedule and personnel are mostly needed to be able to compare with actual schedule and personnel, in order to identify lack of available resources as well as delays in schedule quickly. The amount of proposed requirements tells about the progress of the product definition. The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both on the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Accordingly, defect metrics tell both of the progress of testing, as well as maturity of the product. A set of metrics (Effort, Size, Cost, Velocity, and Productivity) can also be found in the table. These metrics are proposed to be measured by any given iteration of the project because they provide an indication of the project progress. Thus, all reasons for indicated deviations should be carefully studied and analysed. All of the metrics introduced before are proposed to be analysed together with other metrics results in order to gain comprehensive picture of the status.

While developing PRISMA tool integration solution, PSW, project's information and knowledge needs were researched more detail. The needs were also discussed within industrial partners of the project. The aim was to define set of metrics – and their visualised examples – that selected to been implemented on the PSW for demonstrating them in PSW solution. The most of the metrics were

same that were successfully used in industrial environments. Those metrics are introduced in Paper V, and, in addition, the visualised examples and experiences of the selected metric set have been presented in Paper VI. In Subsection 3.2.3, it was illustrated how metrics set can be shown as a dashboard view providing comprehensive visibility into project's progress (Figure 7). In this subsection, each of the dashboard metrics – budget status, testing status, and requirements status – is introduced in details with industrial experiences of their use.

The *budget status* metrics shows actual costs of the project in portion with the agreed budget over a time, visualised in Figure 8.

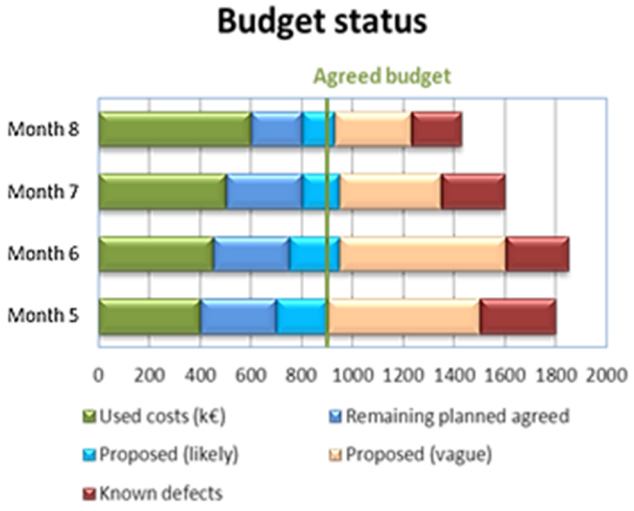

Figure 8. Visualisation of budget status metrics.

The *budget status* metric gives several indicators of estimated prospective costs in each month. The bars summarise amount of costs for the month, and each bar is composed from five different cost-related data. The first block (green) describes actual cumulative costs of the project. The agreed budget for the project is shown clearly as a green line in the middle of the graph. The second block (blue) describes remaining planned cost based on effort estimated for requirements that have been accepted for implementation but not implemented. The third block (light blue), in the middle of the bar, indicates proposed cost that can be seen very likely costs for the project. These costs are based on effort estimated for the proposed requirements that are estimated likely to be implemented, for example, a customer will want them. The fourth block (orange) describes proposed but vague costs for the project. These costs are based on effort estimated for the proposed requirements, which the likeliness for implementation is not known. Instead, the fifth block (red) indicates potential costs for the project, so-called 'Known defects' costs. The costs are based on effort estimated to be needed to fix the known

critical, major, or average defects. Thus, the example graph the *budget status* metric in Figure 8 indicates that the project's costs will overrun the agreed budget.

The metric *testing status* combines effort, requirements, and test metrics in a same graph, illustrated in Figure 9.

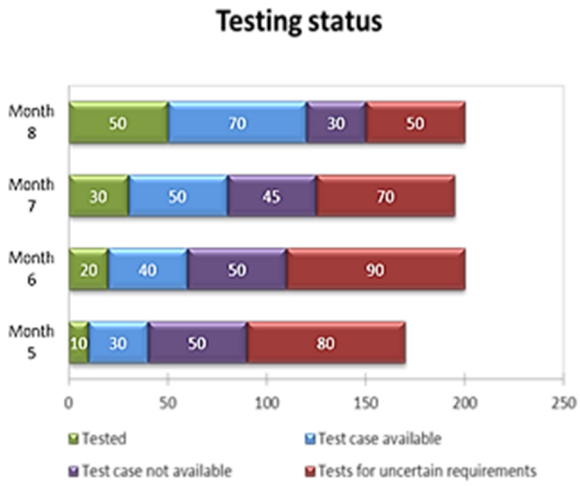

Figure 9. Visualisation of testing status metrics.

The testing status metric visualises the progress of testing phase by collecting data from various phases. The bars in the graph summarise efforts relating to tests in each month. Each bar is composed from four different sums of efforts. The first block (green) describes a sum of efforts for tested requirements. The second block (blue) describes a sum of efforts for requirements for which test case is available, and, accordingly, the third block (purple) describes a sum of efforts for requirements for which test cases are not available. The last, the fourth block (red) is a very proactive indicator, describing a sum of effort estimated for uncertain requirements. Even if 'testing status' shows easily how 'mature' the testing phase is the metric requires other metrics – such as the before introduced metrics: Budget status, progress status, and requirements status – make conclusions based on the data.

The metric of *requirements status* combines the amount of planned effort with status of requirements' implementation over a time in the same graph, illustrated in Figure 10.

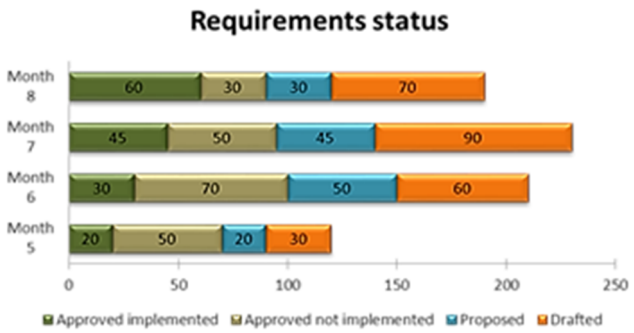

Figure 10. Visualisation of requirements status metrics.

The bars summarise the amount of planned effort for the month. Each bar is composed from four different data relating to identified requirements as follows. The first block (green) describes a sum of planned efforts for all implemented requirements. The second block (grey) describes a sum of planned efforts for approved but not implemented requirements. The third block (blue) describes a sum of planned efforts for proposed requirements and the last block (orange) shows a sum of planned efforts for drafted requirements. It is important to note, that the planned effort is used constantly, even for implemented requirements. This is due to keeping the baseline in order to enable comparing project situation over time, i.e., to be able to see the project trend with respect to planned work. The planned effort may be updated for the requirements during the project, if a new baseline is created. This information is then used together with the actuals, to see how well the planning has succeeded to help learning to estimate better.

The visualised metric ‘requirements status’ indicates several status information but also trend lines relating to requirements implementation, and is focused on showing the uncertainly of the project, for example how much more work maybe dedicated to be implemented in the project. In the example graph, a good signal is that the sum of planned efforts for implemented requirements seems to increase over time while the sum of planned efforts for approved, but not implemented requirements, seems to reduce. However, the sums of planned efforts for proposed and drafted requirements are still quite large in the month 8, especially, while comparing them to the sums of planned efforts for approved requirements. This indicates that the project is in the beginning phase rather than in the ending phase. However, the interpretation needs other metrics information, such as testing status or progress status to make any decisions.

During the PRISMA case studies, industrial experiences of using the metrics were gathered. Industrial partners pointed out that budget metrics give a clear understanding in the actual budget consumption, but, in general, they are poor in predicting budget consumption for the remainder of the project. The *budget status* metric suggested allows for trend analysis and by that extrapolation to the future, resulting in better prediction of the budget consumption for the remainder of the

project. This was mentioned to improve the projects' and the management's insight into the project and enable them to take required measures in a timely fashion, as appropriate. However, the metric requires the project team and stakeholders to agree upon a 'definition of done' which can be very difficult, and even more so if the accepting and implementing parties are different entities or located in different sites. The *testing status* metric provides effective means to get early insight in the status of the product by the end of the project. Moreover, the test-status trend analysis helps to initiate timely measures to work towards an agreed project result. The case study partners pointed that the metric can really improve the insight of project, management, and customer in the status of the product-under-construction and better understanding of what could be expected by the end of the project. The *requirements status* metric was also seemed very promising. They mentioned that in practice the current projects lack insight into the satisfaction of requirements. This lack of insight concerns both the actual status of implementation of the requirements, as well as the expectation: 'Up to what level the project will be able to satisfy its requirements, and if not, what are measures to accomplish that?' The (leading) indicator as proposed in the metric seems to be a good answer to this problem.

Concluding experiences can be highlighted that the interpretation of project's comprehensive status needs various metrics information (like requirements status, progress status, testing status, and budget status metrics) for making conclusions based on the data. Because the metrics need to be easy to read and interpret, it is highly recommended to visualised metrics as a dashboard way for providing effective support for management. This makes possible to interpret and make decisions based on the measurement results so that also certain distributed development implication is taken into account. For example, when interpreting subjective metrics like effort estimation differences between backgrounds of the people (cultural or work experience) may typically influence the results.

Further, collaborative product development forces to pay attention to management and controlling activities for creating awareness of distributed activities within the project and over the projects in an organisation. During the PRISMA project a case study focusing on creating and improving project monitoring, controlling, and reporting practices to attain better coordination between and within the distributed projects as well as to build organisation wide awareness of the status of the project portfolio (Paper III). The following reporting issues were decided to consider with improvements actions in the case:

- Establish monthly synchronization meetings, where the project managers and line managers are synchronizing their activities. The meetings are organised mostly as face-to-face meetings; however, some project managers need to attend the meeting also remotely (i.e., virtual meeting practices are required)
- Request that all project documentation should be updated regularly
- For the project follow-up meeting, establish new monthly reporting templates

- Reporting templates should contain items related to tasks done, open questions, problems, and outlook
- Establish a new project follow-up template into the project portal, so that it is easy for everybody to follow the project portfolio status also remotely from any location where our projects are carried out.

Although, the above practices may seem quite general, they can actually be laboured, time-consuming, or even difficult to carry out during collaborative and hectic projects' situations. Especially, if there are not defined and agreed practices or templates to be followed. The case study showed (Paper III) that the literature and industrial solutions presented in SameRoomSpirit Wiki were relevant even to organisation that was not purely focused on SW development. The results also pointed out that project follow-up and controlling activities improve transparency, e.g., of the project status, between and within the projects through the whole organisation. One of the major findings of the case was that unnecessary work had reduced even 45%. This huge improvement was due to the new implemented reporting practices as well as partly due to other improvement actions made in the unit at the same time (e.g., improving engineering and sales tools, improving process descriptions and document templates). Positive finding was also that project monthly follow-up meetings had been kept regularly each month and reports had been filled in by the project managers into the implemented tool after the implementations. Additionally, the visibility of the status of the customer projects had been experienced to increase remarkably. Although the reporting efforts of the project manager seemed to increase, it also provided them with a communication channel to inform the business management where their efforts were needed the most.

3.3 Benefits of dynamic measurements

In this section, the benefits of dynamic measurements are introduced in relation to the identified measurements challenges in GSD. As introduced before the dynamic measurements have been enabled via the use of automated and real-time indicators with consolidated information via visualised dashboards that were generated by the tool integration solution, PSW. The proposed solution is a proof of concept that was utilised for proving eligibility of the measurement-based management concept itself, for gathering industrial experiences and validating the results as well as for studying further fields of research and development activities in the context of GSD projects.

The dynamic measurements offer and generate a plenty of benefits for their utilising organisations. In the dynamic measurements, metrics, used databases, and analysed results of measurement data are created, updated, and changed dynamically accordingly to managerial needs of projects and demands of various stakeholders involved the collaboration. It has also required that metrics data is collected and analysed continuously from various tools and databases, even from

stakeholders' databases, and measurement data is analysed and visualised for easy to read format. In GSD, this process should be made as easy and as effortless as possible, so measurements automation is needed. Further, it has been pointed out that dynamic measurements shall allow to using companies' legacy tools. This kind of solution, measurement-based management framework, makes possible that information of project progress is up-to-date and the right information is offered to the right persons in timely.

The relevance of measurements needs always careful planning and definition. The measurements have to be focused on the most critical activities and the most important information needs during the collaborative DSD. During the PRISMA project, there were investigated needs for development tools and processes that require more support in GSD. The results showed that needs focused on requirements capture and review processes, traceability needs, testing processes and project management and controlling tasks (Eskeli and Maurolagoitia 2011). Accordingly, Jones (2004) argue that project management have been proved to be the factor that tends to push projects along either the path of success or the path of failure. The most critical activities for successful projects were reported to be project's planning, estimating, change control, and quality control activities. These kind of critical activities and processes need to be supported and managed by relevance metrics in GSD. During the PRISMA project the metrics set that was successfully used in GSD were studied, identified, and introduced in Paper V. Those metrics were aimed especially to provide means to respond proactively to potential issues in the project. Based on industrial experiences of using the metrics (Paper VI), they were meant to be used as a whole, not interpreted as single information of project status. In GSD, the relevance of measurements is better seen by creating well-planned combinations of metrics and visualised measurement results as a cockpit manner.

In the dynamic measurements, the metrics can be continuously updated or even changed if needed. The updating includes also visualisation updates, for example, there can be need to change combination the metrics that are selected to the same visualised graph, or graphs' limiting values need to be changed. In addition, changes in collaboration partners require updates for selected metrics or their combinations. Thus, the metrics set is good to keep as compact as possible because the smaller metric set is easier to adopt, and to update when needed. Even if in GSD the changes are almost continuously needed in metrics, metrics combinations, or visualisation, the benefits are remarkable from viewpoint of project controlling and management. The dynamic measurements make possible to produce real-time, accuracy and focused information to each nominated stakeholders such as project manager, test manager, programme manager and site manager, for example. Based on industrial experiences (Paper VI) the measurements and metrics in GSD should provide online information during the projects, in order to enable fast reaction to potential problems during the project. There were highlighted that one important reason for measurements was that they offers proactive views for development activities, e.g., by introducing 'early warning' signals for the project management. In addition, the metrics were seen as

a ‘balanced score card’, on which management can take the right measures, balancing insights from time, effort, cost, functionality (requirements) and quality (tests) perspective, for example.

In GSD, communication and training relating to used tools or agreed practices are seen very important activities also from viewpoints of project controlling and management. There need to be ensured that agreed tools or data classifications are used in a uniform way by all partners. Missing data or mistakes in work practices can lead to wrong signals from the project, wrong interpreting the metrics, and so even to wrong decision-makings in the project. However, in dynamic measurements, the visualised indicators themselves create means for checking the correctness of gathered data. In practice, most deviations and missing data are easily seen from visualised graphs. Hence, the dynamic measurements can also make visible communication or training needs in collaborative work.

In GSD, the measurements and metrics should indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. The dynamic measurements create real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort. This kind of measurements offers precious benefits for management. In fact, the dynamic measurements make possible to avoid or reduce ‘waste time’, for example, there are possible to re-allocate resources or re-scheduled tasks based on actual needs indicated by metrics. There can be difficult tasks that need more resources or some specific competences, or respectively, the signals can indicate that personnel can be relieved of their tasks. Thus, project management can be made more effective by re-planning project’s tasks. This kind of re-planning can influence on project’s lead-time very positively.

The dynamic measurements create benefits for organisation management level, too. Automation makes possible to utilise measurements over the projects. The case study (Paper III) showed that project follow-up and controlling activities improve transparency, e.g., of the project status, between and within the projects through the whole organisation. For example, the reporting efforts of the project manager increased during the case, and provided them with a communication channel to inform the business management where their efforts were needed the most. Even if, the case study was focused on boosting projects’ controlling and monitoring activities in collaborative production, the results indicate improvements also at project-portfolio management level.

The GSD-related challenges in current measurements practices were summarised in Subsection 2.5.2. Then the requirements for dynamic measurements derived from these GSD-related challenges were introduced and described in Table 5, Subsection 3.1.2. Table 8 introduces how tool integration solution with dynamic measurements can tackle to those challenges and requirements, and what kinds of benefits the dynamic measurements provide stakeholders in managing of GSD projects.

Table 8. The benefits of the dynamic measurements in GSD.

The benefits of the dynamic measurements
C1: Relevance of measurements in relation to project progress The generated views and real-time indicators can be produced based on needs of various roles like project manager, test manager, etc. in different GSD settings. From the viewpoint of the measurements relevance, customisation is a key advantage. For example, the metrics can be defined together with stakeholders ensuring that the importance of the generated metrics has been agreed upon.
C2: Extra work budgeting – metrics design/selection & data collection Even if metrics design/selection needs additional work within the tool integration solution, the work can be moderated with the proposed set of metrics. However, the proposed tool integration solution enables dynamic measurements in GSD. The solution enables to define new metrics by utilising measurements data gathered via the integrated tools. In addition, measurement data is automatically collected after the metrics and views definition: it reduces the amount of extra work for data collection substantially.
C3: Data-reliability issues caused by tools The framework allows a configurable set of development tools and their versions to be tailored for generating real-time indicators of individual project needs. In GSD, the benefits of using the legacy tools are major: the cost of investment of new development tools is typically too high. In addition, people are familiar with their used tools and technology and they tend to resist if they should change their working platform; it would take additional time and extra work, too.
C4: Data-reliability issues caused by human beings The framework cannot affect cultural differences; those divergences should be understood by managers nominated to the tasks. However, the automation in data collection and producing real-time indicators minimises data-reliability problems in these actions. If data must be collected manually, it can induce frustration, especially, if measurements are not understood as relevant for a person's own work. Manual collection can also affect errors or mistakes while transferring data from one tool or form to another.
C5: Extra work budgeting – metrics interpretation Dynamic measurements require data integrations from various tools and sources, and then they are used to construct of visualised graphs and to consolidate information into easy-to-read dashboards. Thus, the dynamic measurements reduce the effort for creating the material and make the analysis and metrics interpretation in GSD easier.
C6: Training needs Dynamic measurements automate data collection for metrics and so reduce the manual gathering and transferring of measurement data. Thus, in this way it also reduces a need for training. However, there will always be training needs since cultural differences and agreed work practices have to be taken into consideration in GSD.
C7: Measurements affects to behaviour Dynamic measurements automate partly, measurements actions and can 'hide' measurements or actual metrics gathered; however, they cannot prevent that measurements can be affected by people's behaviour if metrics are inappropriately selected or designed poorly. The truth is that it is difficult to avoid affecting people's behaviour: 'You will get what you are measuring'. The hiding of measurements may have a positive feature for avoiding problems.

C8: Metrics ethics This challenge is merely a management-level issue that cannot be resolved by any external solution.
C9: Responsibilities and roles are unclear Dynamic measurements, as such, cannot clarify this kind of confusions totally. Instead, it can promote clarifying process essential ways: while defining metrics and creating views it makes visible the needs for metrics data. In addition, unclear responsibilities or roles relating to actual measurements tasks like data gathering, transferring, analysing, and reporting have to be clarified during the process. Further, the dynamic measurements can indicate troubles of unclear assignments via visualised and easy-to-read dashboards.
C10: Continuous changes Dynamic measurements are essential while increasing transparency and creating real-time views between partners in GSD. Dynamic measurements enable integrations of measurements data from various tools and databases as well as consolidate information into easy-to-read dashboards.

Based on Table 8, it can be summarised that attaining measurements is always a knowledge-intensive action, and so efforts by human beings are required. In particular, metrics design and/or metrics selection as well as metrics visualisation (and metrics combination) have to be carefully planned, and thus require some extra work. In addition, metrics interpretation always needs the investment of individuals. However, it is possible that the amount of work required to be done by human beings can be reduced or minimised with the proposed tool integration. The tool-integration solution enables dynamic measurements by creating a new and indispensable approach for measurements and metrics in GSD. Dynamic measurements can help with most challenges related to measurements and metrics in collaborative and distributed software production. In dynamic measurements, metrics can be defined or updated in an agility manner, based on the needs of each project and the demands of each project's collaboration settings. Moreover, metrics data are automatically and continuously collected and analysed from various tools and databases, even from stakeholders' databases. Thus, dynamic measurements can be used to overcome most challenges in current measurement practices and those encountered in distribution and collaboration settings. Traditionally, the metrics were mainly defined once per project, at the beginning of the project. However, there are needs to define, update, or change metrics demands of development settings. In addition, the metrics should provide visibility of the stakeholders' progresses and results, and they need to be analysed and visualised in an easy-to-read format. Dynamic measurements crucial while increasing transparency and supporting project management during collaborative DSD.

As introduced earlier, in GSD, the metrics should be such that they provide online and accurate information during the projects to enable fast reaction to potential problems during the project. It is also important that metrics provide indicators that are easy to read and interpret, and that the information provided offer visibility for all development actions, even over sites and stakeholders in

GSD projects. The research provided the potential metrics set that was successfully utilised in industrial GSD projects (Paper IV), as well as the industrial experiences of dynamic measurements that were produced (Paper VI). Even if the proposed metrics are quite similar to those encountered in single-site development, in GSD projects, actual measurements data items are stored to distributed databases via various tools by different partners. Thus, manual measurement data gathering is experienced as a difficult and even time-consuming task. In GSD, this can lead to managing projects more based on intuition than on facts. Management is grounded in trust and on verbal and written documents informed by partners. Thus, dynamic measurements are necessary to manage GSD projects effectively.

4. Original publications

This section introduces the research performed in the attached publications. Each publication gives answers or clarifies the research questions of the thesis from several perspectives. There are six publications included. Their perspectives drawn from in the thesis and relations to the research process have been introduced and clarified in Subsection 4.1. The topics of the publications are the following:

I Survey of lessons learned by participants of distributed software development

II Building and sustaining measurements practices and processes in a SME (results of a case company, namely Solid)

III Experiences of boosting the controlling and monitoring activities in collaborative production (results of a case company, namely ABB)

IV Knowledge-related challenges and solutions in GSD

V Metrics in distributed product development (results of two case studies, namely Philips and Symbio)

VI Metrics and measurements in GSD (results of two case studies, namely Philips and Symbio).

4.1 Introduction

The thesis covers three different research perspectives: 1) Metrics & measurements, 2) project management, and 3) GSD. The research perspectives have been examined by the author since the year 2000. The topics have been researched during various research projects at the VTT Technical Research Centre of Finland. The author has taken part in large research groups, which have provided the research results from the theoretical approaches of her research interests during the projects. The research projects have enabled the researcher to perform several literature studies relating to the each topic and have offered case study settings, which have arisen from the needs of the industrial partners

involved in the projects. Thus, each perspective has been examined with great subtlety, thus enabling the integration of the research results of each perspective into a newly constructed innovation about the measurement-based management of GSD projects.

Papers I–IV provided a large theoretical background, knowledge, and research results of industrial need and experiences in the context of the research topics. Paper I introduces the results of the large survey of lessons learned by participants of distributed software development. The paper focused on GSD challenges and touched on measurements and project management practices. Paper II discussed the experiences of an industrial case study where metrics and measurements were implemented to practical actions – measurement-data-management framework – in the SME. The paper focused on metrics-and-measurements perspectives, providing an example of successful measurements framework built by the industry. The paper considered metrics and measurements in a quite traditional manner: partly from the viewpoint of software process improvements inside one organisation. The paper focused on describing how to build and sustain the practical measurements environment in the SME. Even if the paper is not focused on the perspective of project management, it points out that measurements provided benefits to project managers when automated and visualised metrics were produced.

Paper III focused on the project management perspective. The paper described how project controlling and monitoring activities could be boosted in a case company environment. This paper did not concern actual metrics or measurements practices but it did report on project managers' needs to control and monitor projects during collaborative, distributed work. The case study was focused on globally distributed product development of software-intensive systems not on software development projects. However, the proposed solutions utilised in the case study were generated based on literature studies of controlling and managing collaborative and distributed software development projects. Thus, the results were included in the thesis. The results showed that project follow-up and controlling activities improve transparency (e.g., of the project status) between and within the projects through the whole organisation. In addition, the case study indicated improvements at the portfolio-management level, even if the case study was focused on boosting projects' controlling and monitoring activities in collaborative production. Then Paper IV discussed knowledge-related challenges and solutions in GSD, as knowledge management and transfer were understood to be very important factors during human-intensive software development. The paper focused on knowledge-related challenges that need to be understood and addressed in order to enable the success of GSD projects. Hence, the presented challenges and solutions covered included metrics and measurements as well as project management perspectives from the GSD viewpoint.

Figure 11 illustrates how these first four publications were situated in the triangle of research perspectives during the research process. These four publications, with large literature studies performed during the research process introduced in Subsection 1.2.2, addressed the research question Q1 (*What are the*

main measurements and metrics related challenges faced by companies when managing GSD projects?). In addition, the publications partially addressed Q2 (*What kind of means and solutions can be found to respond to these identified challenges?*). Especially, Paper IV addressed Q2, even if merely from the knowledge-management perspective.

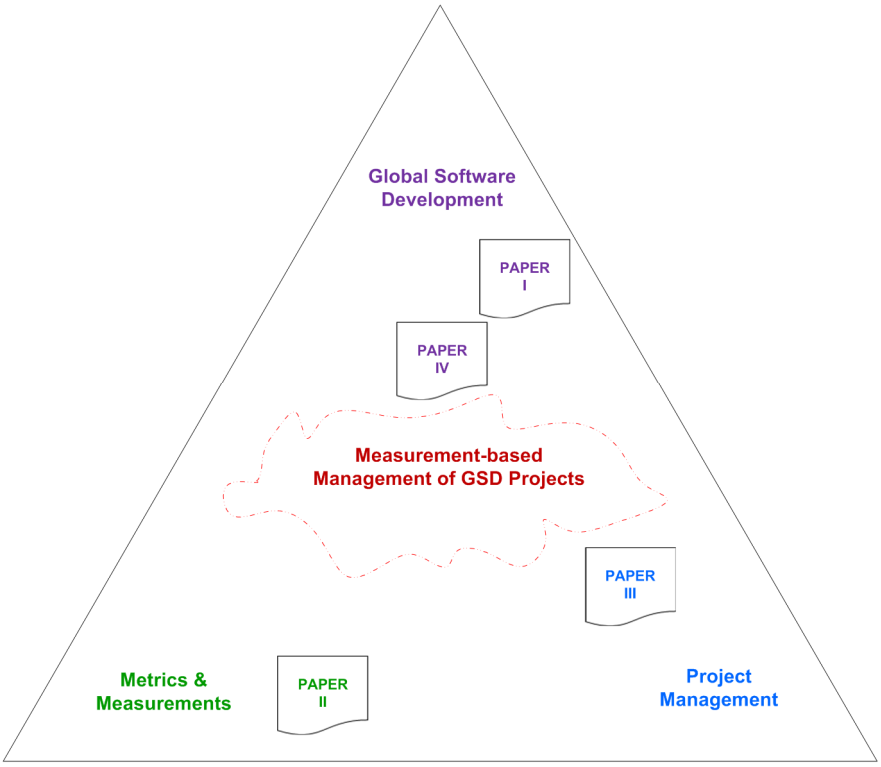

Figure 11. Publications in the triangle of research perspectives.

Paper V introduces the metrics that were successfully used in industrial practice in distributed software development. The paper addresses Q2, completing the results of Paper IV, from viewpoint of metrics and measurements. In addition, the paper provides the first insights for measurement-based management of GSD projects.

Then, Paper VI widened the examination of project management experiences of metrics and measurements in GSD. The paper addressed the research question Q3 (*How measurement-based management are implemented in GSD projects?).* The research results of the paper were utilised while implementing the proof-of-concept framework of measurement-based management for demonstrating and validating research innovations of measurement-based management of GSD projects. The summary of the research process and its main outputs are shown in Figure 2, in Subsection 1.2.2.

4.2 Paper I: Lessons learned by participants of distributed software development

The publication summarises results of the survey that was conducted as a part of Knots-Q project (Knowledge-centered tools and methods for software process quality improvement). The purpose of the survey was to gather and share lessons learned to understand distributed software development better, by identifying the most problematic areas and gathering practical knowledge and examples of the problems experienced along with the potential solutions that have been developed and tested by developers.

The article shows that project management activities such as a careful execution of project start-up activities, detailed planning (splitting tasks, schedule, and deliverables), and exact determination of common rules, responsibilities, etc. can greatly contribute to a successful implementation. In addition, the article pointed out problem areas and concrete lessons learned that should be involved while managing distributed software projects. Being aware of possible pitfalls and potential risks, the project manager should therefore be in a better position to successfully plan and execute projects in the distributed software development environment.

The author participated in a literature study that was as a basis for creating the survey. The web-based questionnaire included both the content and technical issues. In addition, the author was responsible for analysing the responses. The author participated actively during the writing process as a co-author of the paper. The author was also a project manager of the Knots-Q project.

4.3 Paper II: How to build and sustain a measurement data management environment in a SME

The paper introduces the process while building a measurement-data management framework in a SME. The paper shows, via an industrial case, that automated data collection with scripts for visualising results and an Intranet-based measurement environment enable data analysing and process improvement actions in a case organisation. Even if improvements are typically made after the process (as traditionally in process improvements were taken), the case showed that in a SME organisation, corrective actions could be done flexibly during the project work, too. In addition, the paper presents that root cause analysis for achieving deeper knowledge of current practices and processes is possible as the historical measurement data is available. However, these kinds of activities require extra effort and resources, and so these activities are usually passed in a SME.

The author was a main author of the paper. The author conducted the literature study of the measurement elements relating to quality of software products and processes, and proposed the MDM framework for application in the SME environment.

4.4 Paper III: ABB experiences of boosting controlling and monitoring activities in collaborative production

The publication describes a case executed at ABB (http://www.abb.com/) during the ITEA PRISMA (2008–2011) project. Distributed collaborative product development requires that attention be paid to management and controlling activities for creating awareness of distributed activities within the project and over the projects in an organisation. The paper introduces ABB experiences in boosting globally distributed project management activities by integrating those actions to the portal of the company. The case focused on creating and improving project monitoring, controlling, and reporting practices to attain better coordination between and within the distributed projects as well as to build organisation-wide awareness of the status of the project portfolio.

The case showed that the implemented reporting practices together with other improvement actions, such as enhanced process descriptions, practices, and engineering tools, reduced the unnecessary or free work done at ABB by 45%. This successful result was achieved by careful current-state analysis, where the requirements and goals for improvement actions were defined. In addition, the improvement actions were defined by discussions with essential interest groups as well as by considering the best practices and research results from the literature.

The paper points out that project follow-up and controlling activities improve transparency (e.g., of the project status between and within the projects through the whole organisation) in the globally distributed product development environment. Project management practices play an important role in distributed product development. In addition, the monitoring and reporting activities should support and improve transparency with real-time and visualised indicators within the project and over the projects in an organisation.

The author was the main author of the paper. In addition, the author was a research partner and facilitator during the industrial case.

4.5 Paper IV: Knowledge related challenges and solutions in GSD

The publication introduces knowledge-related challenges in the GSD that need to be understood and addressed in order to enable the success of the GSD projects. The role of knowledge and knowledge engineering is crucial in software development projects, but it is even more important in GSD because of the distance and cultural aspects.

This publication points out that a successful distributed software development project requires both structured and disciplined software engineering and knowledge-management solutions. Communication management and the utilisation of effective substitutes for face-to-face communication have an important role in GSD, to ensure knowledge sharing. For example, ensuring the availability of information during the project to all of the parties is essential for a successful project.

From measurement and project management viewpoints, it is important to understand the nature and demands of the GSD; for example, in a distributed development project, a significant amount of effort is required for up-front planning and follow-up activities to manage a project successfully. The distribution makes the project progress more difficult to estimate and control because of the decreased visibility. In addition, a manager 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. 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. In addition, knowledge engineering was recognised to hold a vital role in the analysis and interpretation of the measurements. In order to make correct conclusions from the data knowledge sharing, the lessons learned have to be taken into consideration.

The author is a co-author of the paper and is responsible for incorporating the knowledge-engineering viewpoints to GSD; whereas, the author Päivi Parviainen was responsible for analysing the challenges identified in over 50 case studies. The author focused on case studies concerned with measurements, metrics, and project and quality management issues in GSD projects.

4.6 Paper V: Metrics in distributed product development

The publication describes a set of metrics that was successfully used in industrial environments during distributed product development. The main purpose of the paper is to share knowledge by offering a set of essential metrics with concrete experiences of their use. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. The paper highlights, based on industrial experiences, that the metrics should be such that they provide online information during the projects, to enable fast reaction to potential problems during the project.

Metrics are seen as important activities for successful product development, as they provide means to monitor the project progress effectively. However, globally distributed development generates new challenges and difficulties in terms of measurements. For example, the gathering of the measurements data can be problematic because of the use of different development tools or their versions, because work practices with related concepts can vary by project stakeholders, or because the reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations.

The paper focuses on describing a set of metrics that has been successfully used in industrial practice in distributed product development. These metrics are aimed especially to provide means to react proactively to potential issues in the project, and they are meant to be used as a whole, not interpreted as single information of project status. Moreover, they are easy to capture and can be quickly calculated and analysed at regular intervals.

The author was the main author of the paper. In addition, the author was a research partner and facilitator during the industrial case.

4.7 Paper VI: Metrics and measurements in global software development

The publication is focused on describing a set of metrics with visualised examples and experiences of their use. The main purpose is to introduce the selected metric set from the viewpoint of their proactive role in decision making during globally distributed software development. The chosen metrics indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD projects, where information of project status is not readily available but requires special effort, distributed over sites and companies.

The metrics and discussion in the article are based on GSD improvement work carried out during several years, in several research projects. In this paper, the first ideas of GSD specific metrics are presented based on the common challenges in GSD practice.

Even if most of the introduced metrics are similar to those of single-site development, their collection, and interpretation need to be taken into account concerning the GSD aspects. This publication focuses especially on the experiences of two companies, Philips and Symbio. One of the most important reasons for choosing the proposed metrics was their provision of early warning signs, to respond proactively to potential issues in the project. This is especially important in distributed projects, where tracking the project status is needed and proactively more complex. The balancing insights, from time, effort, cost, functionality, and quality, are also seen as a very important aspect.

The author was the main author of the paper. In addition, the author was a research partner and facilitator during the industrial case.

5. Discussion

In this section, the results of the thesis are discussed and evaluated. First, in Subsection 5.1, a description of how the selected research methods were applied in the research is given. Then in Subsection 5.2, the results of the research are discussed according to theory (5.2.1) and practice (5.2.2). Finally, in Subsection 5.3, the limitations of the research are introduced and discussed.

5.1 Validity of the research

The main research methods used in this thesis were literature studies and case studies within the design-science research approach. Because of the broad theoretical context of the thesis, several literature studies were conducted as a part of certain research projects introduced in Subsection 1.2.2. The main results of the literature studies, such as published handbooks – MIKKO Handbook, Collaboration Handbook, and SameRoomSpirit Wiki-based handbook – were reviewed by internal and external reviewers during and after the projects. According to Easterbrook et al. (2008), the major weakness of the case study method is that the data collection and analysis is very open to interpretations affected by the researchers' biases or cultural backgrounds, for example. To address this challenge, industrial case studies were led by the industrial participants themselves and the results were collected, documented, and reported by the researcher. All documented results were discussed and reviewed with the industrial partners involved the cases. In addition, the construct validity of the main research results and case studies utilised in the thesis were ensured by submitting results to be reviewed and published in the various, carefully selected publication forums introduced in Subsection 4.1.

Commonly used criteria to evaluate the validity of research (Easterbrook et al. 2008; Yin 2009) include construct validity, internal validity, external validity, and reliability. Construct validity centres on whether the theoretical constructs are interpreted and measured correctly. Internal validity centres on the study design and particularly on whether the results really do follow from the data. External validity focuses on whether claims for the generality of the results are justified. Reliability focuses on whether the study yields the same results if other

researchers replicate it. The thesis contributes to understanding the phenomenon called measurements-based management of GSD projects. In practice, the design science research provides a wide perspective for the research, and it covers the previously mentioned general requirements of construct, internal, and external validity as well as reliability for case studies in the social sciences. In fact, Wohlin et al. (2003) recommend that design science be applied also in empirical software engineering. Thus, in the thesis, the design science research guidelines introduced in Table 9 are used while discussing the validity of the research in more detail.

Table 9. Design science research guidelines (Hevner and Chatterjee 2010).

Guideline	Description
1: Design as an artefact	Design science research must provide a viable artefact in the form of a construct, a model, a method, or an instantiation.
2: Problem relevance	The objective of design science research is to develop technology-based solutions to important and relevant business problems.
3: Design evaluation	The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods.
4: Research contributions	Effective design science research must provide clear and verifiable constructions in the areas of the design artefact, design foundations, and/or design methodologies.
5: Researcher rigor	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact.
6: Design as a search process	The search for an effective artefact requires utilising available means to reach desired ends while satisfying laws in the problem environment.
7: Communication of research	Design science research must be presented effectively to both technology-oriented and management-oriented audiences.

The design science research guidelines (Table 9) are used to demonstrate the validity of the research in the following paragraphs.

Design as an artefact. Because the thesis covers three different research perspectives (metrics & measurements, project management, and global software development) that were examined in various research projects, introduced in Subsection 1.2.2, several artefacts were produced. The main instantiation from a metrics and measurements viewpoint from the MIKKO project was the MIKKO Handbook, Comprehensive collection and utilisation of software measurement data, focusing on organisational and project level measurement utilisation. The

MERLIN project focused on embedded systems engineering and software engineering technologies from a collaboration perspective, and it provided, as a main result, the MERLIN Collaboration Handbook, which summarises the main research results of the project. The purpose of the handbook is to support operational collaborative development (e.g., it helps companies to take care of all of the critical aspects during various phases of the collaborative work, providing potential solutions to address these challenges). The solution were based on literature, especially for management and support practices (including measurements practices), and on the collection of the best practices from the MERLIN industrial partners via focused interviews on selected topics. Further, during the PRISMA project, the MERLIN Collaboration Handbook was further developed with a widen context, being freely available from the Intranet as SameRoomSpirit Wiki. The research projects offered environments where metrics and measurements as well as project management practices could be investigated in detail. This was done via literature surveys and case studies, which were introduced and published as the attached papers. This large study enabled the researcher to integrate the previously mentioned research perspectives with the main construction, the tool integration framework, PSW. The implemented tool integration solution has been discussed and published in (Pesola et al. 2008) and in (Eskeli et al. 2011). PSW was utilised as a proof of concept of measurement-based management of GSD projects, and it made it possible to demonstrate dynamic measurements during collaborative and distributed software development and to collect feedback and experiences of the construct introduced in Section 3.

Problem relevance. The industrial partners of both the MERLIN and PRISMA projects actively participated in inventorying their challenges and problems in GSD as well as in providing and evaluating new solutions addressed to them. In total, 52 industrial case studies were carried out to evaluate the solutions identified for the challenges revealed in the industrial inventories, for example. These case studies and research processes were introduced and discussed in (Parviainen 2012). In addition, project management and measurements-related challenges arose from industrial needs. In fact, the main results of the PRISMA projects, the SameRoomSpirit Wiki, and PSW tool integration solution were developed in tight co-operation with the industrial partners involved the project. There were also case studies of utilising those constructions from where gathered feedback and experiences were utilised in their further development work. Thus, the developed construction, the tool integration framework with implemented metrics, was very important and relevant to the identified business problems. In fact, the problems and challenges addressed during the industrial projects were defined by the company themselves, not by the author.

Design evaluation. The thesis points out that measurement-based management of GSD projects is possible and very important, and it provides many benefits for all stakeholders in the collaborative setting. For demonstrating the construct, a proof of concept of the technical implementation during the PRISMA project was demonstrated. Because the construct contains the technical implementation (tool integration solution, PSW) with proposed metrics as well as viewpoints for

measurements, evaluations were performed via many industrial cases and from various aspects. First, the tool integration solution was designed, developed, and evaluated in several phases. The first idea and implemented concept of tool integration was developed and evaluated during the MERLIN project (Pesola et al. 2008). During the PRISMA project, the tool integration was further developed. The industrial partners utilised and evaluated the tool integration solution in their real world distributed project or in their own multisite software development projects (Eskeli, Maurolagoitia, Polcaro 2011). In addition, the solution was tried within a few demonstrations and research settings during the PRISMA project. The actual set of tools and amount of people involved was dependent on the context and duration of each trial. In fact, those experiments and demonstrations were fixed on testing implementations and functionalities, and so focused on further developing the solution itself. For example, one tool, AgileReq, was integrated based on the company's needs. In fact, one principle for the development work had been that, in GSD, companies have their own practices and tools, and the tool integration solution should not enforce a specific process or tool set. In addition, feedback related to managerial issues and measurements in GSD were gathered during the experiments. In addition, a set of metrics in the tool integration solution were demonstrated. Because the development of metrics or the selection for producing relevant ones for the project management is a challenging task, the metrics development for the tool integration solution was done in close co-operation with the PRISMA projects' industry partners. The selected and implemented metrics were experienced as being useful, and they were successfully used in industrial practice (Paper VI). Actually, the research relation to the tool integration solution covered several case studies in various companies, as well as case experience reports published by others. Thus, they provided multiple sources of evidence. In practice, the case studies were carried out in real industrial projects, and conclusions were reached in co-operation with industrial representatives. The results of case studies were also reviewed by more members from the companies as well as other researchers involved the projects. The results of each case study were also included on Internet-based handbook solution, SameRoomSpirit Wiki.

Research contributions. The main contribution of the thesis is the construct of measurement-based management of GSD projects. The research provided the tool integration solution that was utilised as a proof of concept, while demonstrating and validating the results. In reality, a solution of measurement-based management is depending on various factors such as on used/selected tools, collaboration modes or amount of partners involved, so only one and stabile solution cannot provide. For example, stakeholders have already selected tools that support their way of working or companies have already defined practices that work well and so they are not willing to change tools or practices. Thus, it was necessary to provide a technical implementation that was used as a proof of concept for the construct.

Researcher rigor. As described in Subsection 1.2.2., the research process has been long, and it has involved several research projects. This has enabled several literature studies for perceiving a strong theoretical background for the thesis as

well as finding a cap between theory and practice. In addition, the research projects not only provided industrial environments for case studies but also reviewers for the results, thus maintaining the high quality of the research. All case studies were reported a structured manner and reports were reviewed by researchers and industrial representatives involved the projects. In addition, all main research results introduced in the thesis have been reviewed and published in scientific forums, conferences, and journals.

Design as a search process. The research was performed in numerous different research projects involving industrial representatives and researchers with high and long-term expertise in the research topics of the projects. This made possible constructive approaches for the research questions introduced in Subsection 1.1. The performed research process has been described in detail in Subsection 1.2.2. Several large literature studies were conducted, enabling the researcher to construct innovations that were further studied in detail in the industrial context. The gathered feedback and experiences were processed with further and more focused literature studies. In addition, workshops were arranged with industrial partners for identifying and analysing their problems faced in GSD. Moreover, there was an agreed formal method for reviewing and validating all provided case study and research reports during the projects.

Communication of research. As described previously, industrial representatives of the research partners actively participated in the research process. In addition, there were several public seminars where the main results of the research project were communicated. For example, in spring 2010, the industrial seminar '*Same room spirit in multisite fashion*'[11] were held, where the tool integration solution, SameRoomSpirit Wiki, and various industrial experiences relating to the topics of the PRISMA research project were communicated. These seminars were aimed at technology-oriented and management-oriented audiences. In addition, scientific forums such as conferences and journals were utilised in communicating and publishing the results of the projects. These scientific forums provided external review processes for the results, being one valuable validation method for the results.

5.2 Evaluation of the results

In this section, the research results of the thesis are evaluated from the viewpoint of theory and practice. First, theoretical contributions are discussed in Subsection 5.2.1, by evaluating how the research addresses the research questions introduced in Subsection 1.2. Then, the main practical implications are discussed in Subsection 5.2.2.

[11] http://conference.erve.vtt.fi/srs2010/ Accessed 15.10.2014

5.2.1 Theoretical contribution

The main contribution of this work is to combine separate views, such as measurements, metrics and their interpretations tools as well as challenges and needs for administrative information in the context of the measurement-based management of global software-development projects. The thesis brings together these publications and industrial cases with a proof-of-concept implementation that made it possible to evaluate the findings during the research.

Relating to the first research question, *What are the main measurements and metrics related challenges faced by companies when managing global software-development projects?* Trends in GSD show that the products are being increasingly developed in a globally distributed fashion, where the size and complexity of software-intensive systems are continuing to grow. Management of a distributed product development project is proven more challenging than traditional development is. Because measurements and metrics create useful ways for controlling and managing projects, these activities are emphasised in the management of GSD projects. The benefits and needs for measurements are discussed in Subsection 2.4.1. Even if measurements bring several benefits for an organisation, the fact is that organisations usually cannot allocate enough time or resources to do the measurements properly. There are many problems and challenges that have been identified that have reduced or even to eliminated interest in the measurements in GSD. The measurements and metrics-related challenges, can generally be summarised as 1) the measurements practices, metrics, and tools themselves, 2) challenges caused by people, and 3) challenges from the collaboration settings. In addition, these challenges are strongly dependent on each other. For example, GSD settings affect the current measurement practices by producing new challenges or complicating old ones. The detailed list of challenges is presented and discussed in Subsection 2.4.5. Many of the challenges occurring in a distributed development case are almost the same as those faced in 'single-site' development. However, the distribution makes these challenges more problematic and complicated. In addition, the distribution brings new challenges in terms of measurements and metrics. Most challenges concern the need to get reliable, real-time information (measurements results) as automatically as possible. The needs centred on information that could be used in decision making, especially, as proactively as possible. In GSD, the challenges faced more complicated than those found in traditional one-site development cases, because measurement data items were located in different tools and databases, typically even in stakeholders' databases.

The second research question was *What kinds of means and solutions can be found to respond to these identified challenges?* The introduced measurements needs and identified challenges in measurements practices built a base for searching for and developing solutions for the measurements challenges faced in GSD. These solutions were identified, developed, and adopted from the literature as well as industrial cases during the MERLIN and PRISMA projects. The

research included the study of knowledge management and transfer-related challenges and solutions as well as tools, methodologies, and metrics and measurements–related challenges and their solutions in GSD. The industrial case studies were performed to gain feedback and knowledge of measurements and metrics needs and industrial experiences as well as to use successfully metrics and projects' controlling practices in GSD. The research pointed out that demands for dynamic measurements are substantial: the metrics need to be defined and updated, case by case; they should be followed on a daily based, and they should provide support to several stakeholders in different roles during the collaborative work. In practice, the management of GSD projects requires well-optimised automated and real-time indicators that provide up-to-date visibility of the stakeholders' progress and results. For achieving optimal, well-balanced, and complete dynamic measurements, knowledge-related factors need to be considered.

A set of requirements for dynamic measurements was provided as a research result. The requirements were introduced in Subsection 3.1.2. These requirements were derived from GSD-related challenges in current measurement practices. They clarify the kinds of measurements and metrics that shall be utilised while producing information to manage GSD projects, and how the main measurements or metrics–related challenges can be tackled in GSD. The requirements help when developing solution(s) for the identified needs and challenges in GSD.

To summarise the requirements, it can be easily detected that automation is one important requirement: automation helps to provide reliable, real-time, and visualised, easy-to-read indicators to support decision making in the complex, uncertain, and dynamic environments of GSD. In addition, knowledge management and knowledge-intensive perspectives should be considered while developing solutions to GSD-related challenges of current measurement practices. The dynamic measurements were important while increasing transparency and information sharing during collaborative distributed software development. Dynamic measurements were defined as measurement actions, where metrics are defined or updated based on the needs of each project and the demands of each project's collaboration settings, where metrics data are collected and analysed continuously from various tools and databases, even from stakeholders' databases, and measurement data are analysed and visualised in an easy-to-read format.

The third research question was *How can measurement-based management be implemented in GSD projects?* The second research question provided the main requirements and practical means for dynamic measurements in GSD. Thus, these requirements were used while developing a proof-of-concept implementation of measurement-based management in GSD. The proof of concept provided a validation framework for measurement-based management approach in GSD. It provided the framework where proposed requirements and potential metrics were implemented for gathering industrial experiences and validating the results. The introduced tool integration solution was designed, developed, and piloted during the MERLIN and PRISMA research projects. The proof of concept with demonstrative and piloting feedback was utilised for examining the third research question. The technical implementation included a set of metrics, and it enabled

dynamic measurements and metrics in GSD. One important principle for the implementation of the measurement-based management framework was that solutions should not enforce a specific process or tool set; instead, the tool integration shall be enabled with companies' own practices and tools used in GSD. The research showed that dynamic measurements were enabled via the use of automated and real-time indicators with consolidated information via visualised dashboards that were generated by the tool integration. The research pointed out that the real benefits of the proposed tool-integration solution can be achieved when several visualised views can be read simultaneously. The interpretation of the project's comprehensive status needs a variety of metrics information, such as introduced requirements status, testing status, and budget status metrics together. The results also pointed out that project follow-up and controlling activities improve transparency (e.g., of the project status) between and within the projects through the whole organisation. The measurement-based management can really be utilised while managing and controlling GSD projects.

Project management includes knowledge management activities that need to be taken into consideration, especially in GSD, where partners, contact persons, and tools vary a lot. Tools or automation cannot replace face-to-face communication or offer knowledge that is needed in different collaboration settings. However, measurement-based management can offer views to the progress of partners' project or tasks, and it can also make visible possible problems or challenges in development tasks, and, thereby, it creates transparency between the partners in GSD. Thus, measurement-based management is an important and valuable framework while managing GSD projects. The thesis is focused on globally relevant and important problems in software engineering. Some of the research results confirm 'held beliefs' in reality and this way are scientifically important arguments. Further, the thesis acquires further insights into the phenomenon of measurement-based management of GSD projects, which enrich the current body of knowledge and provide material for further research. This thesis is focused on globally relevant and important problems in software engineering. The thesis provides further insights into the phenomenon of measurement-based management of GSD projects, which enriches the current body of knowledge and provides material for further research.

5.2.2 Implications for the practice

The thesis provided results regarding the practical implications during the case studies as well as more general implications for managing GSD projects. In the following, these two main aspects of the practical implications from this thesis are discussed.

The first implication concerns project-management practices in GSD the use of metrics and measurements therein. In collaborative and distributed software development, the importance of project management is emphasised. In addition, the competencies and skill requirements of GSD project managers are extended.

For example, the continuously increasing complexity of the business requires the project manager to deal with many roles, such as business managers, customer managers, marketing managers, product managers, suppliers, etc. For example, Wu et al. (2009) argued that project management is one of the primary factors to software projects' success or failure. In GSD, a project manager is often far away from the development teams or stakeholders, which creates visibility and project controlling problems. It is also easier to hide problems. Further, Herbsleb (2007) pointed out that the key phenomenon within GSD projects is coordination over distance: the need to manage a variety of dependencies across sites drives the essential problems of GSD. Thus, systematic controlling and status reporting of the project work is especially important in GSD. In practice, measurement and metrics provide important means to do that effectively. Paper II introduced the main elements that need to be considered while building and sustaining a measurement framework in an organisation. Because the case study was conducted in a SME environment and it was merely focused on building a measurement framework for purposes of traditional measurements (not dynamic measurements), the GSD-related challenges have to be carefully studied together the proposed elements. For example, management of GSD projects may require new or changed practices for project planning and tracking because of collaboration, for example, due to schedule dependencies that need to be managed. In addition, it is important that status reporting practices and change management procedures be defined clearly, and this includes reporting channels, decision authorities, and escalation channels. Thus, the dynamic measurements are important while increasing transparency and information sharing during collaborative distributed software development. The thesis included a set of metrics that has been successfully used in GSD projects. The metrics are introduced in detail along with the experiences of their use in the attached publication Paper V. In addition, for achieving optimal, well-balanced, complete dynamic measurements, knowledge-related factors in relation to the team activities need to be carefully examined. Paper IV introduces the challenges that the companies had faced in GSD, and it discusses their knowledge-engineering aspects and presents 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 problems and may lead to the poor sharing of information or a lack of knowledge about what to do. Paper IV provides a cognitive perspective on the challenges faced in GSD, and this way it helps in finding solutions that take into account the knowledge needs of different stakeholders in GSD. It was proved that successful distributed software development requires both structured and disciplined software engineering and knowledge management solutions.

Another implication concerns a practical tool integration solution for providing measurement-based management solutions for managing GSD projects. The thesis introduced the tool integration solution that was utilised as a technical

implementation of the measurement-based management framework. The thesis discussed how GSD-related challenges affected metrics and measurements practices and it also introduced a set of requirements for dynamic measurements. Dynamic measurements concern metrics definition/selection, optional metrics combinations, measurements automation (partially or completely), and metrics interpretation with visualised indicators, for example. In addition, Paper VI describes a set of essential metrics that were successfully used in GSD. Also given, were visualised examples based on experiences for demonstrating their use in industrial projects. The proposed metrics provide early warning signs for a project and so they make possible to react proactively to potential issues in the project. In the thesis, the technical implementation of the tool integration solution was introduced in detail, by offering examples of dynamic measurements and their benefits for project management in GSD. The example solution provides real-time views of development assets and an infrastructure where assets are synchronized and communicated efficiently. The actual workspace was a collaboration and integration portal. The benefits of the proof-of-concept framework are that the solution is vendor independent, and it allows for a configurable set of development tools that can be tailored to individual partner or project needs. The implemented tool integration solution provided a concrete example of the construct of measurement-based management of GSD projects. Moreover, it offered a proof-of-concept framework, where dynamic measurements were studied, demonstrated, and evaluated during the research.

5.3 Limitations of the research

The research results provided were produced within several research projects during the long period. The research projects made it possible to investigate theoretical backgrounds from each research perspective introduced in Subsection 1.2.2. The thesis has been built from several research portions that were provided via industrial case studies. These case studies were carried out in real industrial projects, and the results of the case studies were validated during the research process. The main research results included in the thesis have been published in high-quality scientific forums. The introduced constructs of measurement-based management and dynamic measurements are innovations that were produced by the research results. It has been noticed that, increasingly, commercial tools are being developed to provide support for project management with dynamic measurements in GSD, by developing new product families or expanding old ones; however, those kinds of big commercial solutions are often too ‘massive’ and expensive, at least for small companies. Thus, many companies are forced to develop their own tools or tool integrations to support their GSD projects. For example, many proprietary solutions have been developed for single tasks or project phases that have been identified to be the most critical or the most cost-effective in GSD practices. However, these kinds of tailored solutions are typically aimed to solve single problems (e.g., providing integration for certain development

assets, being tools used in integration). The thesis introduced an example solution that provides support for measurement-based management without depending on certain vendors or tools. The proof-of-concept solution allows a configurable set of development tools that can be tailored to individual partner or project needs. In addition, it demonstrated how dynamic measurements could be utilised in managing GSD projects.

The main research results were reviewed and evaluated during the research projects and in the published papers attached. In addition, the construct of measurement-based management of GSD projects has been evaluated via the feasibility study of the PSW tool integration solution. However, long-term research in real industrial projects is lacking. Thus, further research would address this limitation: dynamic measurements would be implemented in several different distributed software development projects, for example. This would make it possible to examine and measure the effects on dynamic measurements to increase transparency and support decision making actions from viewpoints of various stakeholders in GSD projects.

6. Summary and conclusions

In this section, the results of the research are summarised (6.1), and further research recommendations are discussed (6.2).

6.1 Summary of the results

This thesis summarised six original publications and extended them through the construction of measurement-based management of GSD projects. The technical implementation of the tool integration solution was introduced and discussed for constituting a proof of concept of dynamic measurements in GSD.

In fact, software measurements and metrics in software production have been discussed and studied over several decades because they are commonly understood to create concrete means for monitoring and controlling projects and providing support for projects' decision making and management. The needs for project controlling actions are emphasised in GSD because distributed product development generates new measurement challenges and difficulties. For example, gathering measurements data is problematic because of the different development tools and their versions in use in the project, because work practices can vary by project stakeholders, and because the reliability of the gathered data can vary because of cultural differences, especially, in subjective evaluations. In practice, there are needs to gather the measurements data from multiple sources, such as different tools and databases, even from stakeholders' databases, during GSD projects.

In software production, measurements are understood as an information-gathering process, where measurement data consist of numeric data or a pre-classified set of categories. Because software metrics can consist of several measurement data items individually or in combination, measurements and metrics have been strongly linked to the development tools used during the production. In the literature, many problems and challenges have been identified that reduce or even eliminate all interests to the measurements. However, the amount of GSD-specific literature on metrics and measurements or that even discusses the topic is limited. In fact, some papers and books exist, but they discuss metrics in general or only for specific aspects, such as quality metrics to

support the defect management process. In the thesis, challenges in current measurement practices have been summarised and described in detail from a GSD viewpoint. Further, requirements for dynamic measurements derived from GSD-related challenges in current measurement practices have been introduced and summarised.

This thesis defines *dynamic measurements* as actions where metrics are defined or updated based on needs of each project and demands of each project's collaboration settings. The actual metrics data are collected and analysed continuously from various tools and databases, even from stakeholders' databases, and results of measurements are analysed with visualised indicators that are easy to read. In dynamic measurements, metrics, the databases used, and the analysed results of measurement data can be created, updated, and changed dynamically, based on the managerial needs of each project, its different phases and tasks, as well as the managerial needs of various stakeholders in collaborative work. The construction of dynamic measurements could be demonstrated and validated via the implemented tool-integration solution during the research project. This thesis introduced a technical implementation that was utilised as a proof of concept for the measurement-based management of GSD projects. The example solution made it possible to demonstrate, evaluate, and collect experiences of the construction in real industrial projects during the research. It was reported that the tool integration solution offered better visibility beyond stakeholders' borders as well as into project progress through tool support in communication and project management during the research. In addition, resource management was assessed to be more efficient because of better transparency (traceability of design assets, awareness, etc.) between sites and stakeholders. The thesis concluded that the main results of the proof of concept feasibility studies, measurement-based management of GSD projects, is a very effective way to provide support for project management in the challenging environments of collaborative and distributed software development.

6.2 Future research

The work introduced and discussed in the thesis is based on extensive empirical work, carried out over several years. The actual research work included several literatures studies that were processed from several perspectives: metrics and measurements, project management, and GSD. In addition, knowledge management and transfer were studied in order to expand on the understanding of challenges in current measurement practices as well as creating and addressing potential solutions to them.

The topics of the thesis are very large, and, therefore, there might be aspects or exact problems that have not necessarily been covered in the thesis. However, the thesis covers things that have been seen as important in the case companies involved the MERLIN and PRISMA projects. In addition, literature studies attested that those challenges and needs must be supported during the management of

collaborative projects. The thesis summarised the research results that have been reviewed and evaluated during the research projects. The thesis provided the construct of measurement-based management of GSD projects that were demonstrated and evaluated via the feasibility study of the example tool-integration solution. Because long-term research in real industrial projects is lacking, further research is still needed.

Future research actions could be divided into theoretical and empirical research approaches. From the theoretical viewpoint, future research would include a large industrial survey of dynamic measurements in the context of GSD projects, for example. The thesis covers studies of problems in current measurements and metrics practices with studies of practical solutions developed and proposed to meet those challenges in GSD projects. These studies were carried out within the industrial partners that were involved the MERLIN and PRISMA research projects. In addition, literature examinations were included. As a result, the context of dynamic measurements was defined and tested via an implemented proof-of-context research prototype of tool integration solution. Thus, the future research would focus on measurements practices and metrics in various commercial and proprietary tool integration solutions that provide support with dynamic measurements for managing of GSD projects. This kind of survey could also include different collaboration modes, such as customer-supplier relationships, technology exchange, joint research and development, and in-house distributed development, especially, their influences and demands for dynamic measurements. Further research would also consider challenges and solutions from the viewpoint of various stakeholders or different roles involved in collaboration. In addition, new research fields could enrich the results and create new innovations for the research. For example, regarding the approaches of cross-cultural groups and organisations, how do they affect project management and how do they effect to measurements and metrics or measurement practices in GSD?

From empirical viewpoints, the future research actions would focus on gathering experiences from various kinds of GSD environments, metrics, and projects. For example, dynamic measurements would be implemented in several different DSD projects for providing real industrial environments to examine further the challenges, needs, and potential solutions. One potential option could be to implement the tool integration solution in an industrial product development environment and then gather experiences via action research methods and questionnaires for several years period. Another choice could be build different kinds of infrastructures for experimental research purposes. In addition, adding new types of tools (e.g., for change management or design) to potential tool integration solution would provide new insights for controlling and managing the GSD projects. Also, experiences of new potential GSD related metrics would be gathered, studied, and analysed. For example, metrics focused on measuring the project performance, especially, task and team performance in GSD would be interesting. The potential metrics could be measurements related to time spent idling, e.g., waiting for something, and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when

a team is not performing. All these kinds of experiences as well as experimental methods could elicit new information and new contributions to the theory. For example, experimental research could provide recommendations or restrictions of dynamic measurements in different kinds of organisations or collaboration modes involved. These kinds of experimental research and case studies make it possible to investigate and measure effects on dynamic measurements in GSD.

The thesis summarised the long-term research activities and industrial case studies about the challenges and solutions associated with measurements practices and metrics in GSD projects. The thesis concluded with the construction of measurement-based management of GSD projects that were demonstrated and evaluated via the implemented tool integration solution and the proposed set of dynamic measurements. The feasibility study provided experiences via industrial demonstrations and case studies and hence showed several benefits of dynamic measurements in management of GSD projects. However, long-term research in real industrial projects is lacking. The implemented tool integration environment with dynamic measurements is not extensively used in industrial product development environments and projects. However, the implemented tool integration solution provided a concrete example of the construct of measurement-based management of GSD projects, and it offered a proof-of-concept framework, where dynamic measurements were studied, demonstrated, and evaluated during the research.

The thesis concluded that dynamic measurements are a necessity in the management of GSD projects. The thesis concluded that measurement-based management of GSD projects is a very valuable and effective way to provide support for project management in the challenging settings of distributed and collaborative project work. The thesis pointed out that dynamic measurements provide solutions to the identified challenges of current measurement and metrics practices in managing GSD projects. In addition, several concrete benefits of utilising dynamic measurements in GSD environments have been summarised.

References

Al-Ani B. and Redmiles D. 2009. Trust in distributed teams: Support through continuous coordination. IEEE Software 26(6): 35–40.

Altidor W., Khoshgoftaar T. M. and Napolitano A. 2009. Wrapper-based feature ranking for software engineering metrics. In proceedings of international conference on machine learning and applications, ICMLA'09. IEEE. Pp. 241–246.

Basili V. R. 1992. Software modeling and measurement: The Goal/Question/Metric paradigm. University of Maryland at College Park.

Batagelj V., Bojkovski J. and Drnovšek J. 2008. Software integration in national measurement-standards laboratories. IET Science, Measurement & Technology 2(2): 100–6.

Battin R., Crocker R., Kreidler J. and Subramanian K. 2001. Leveraging resources in global software development. IEEE Software 18(2): 70–7.

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.

Bourgault M., Lefebvre E., Lefebvre L. A., Pellerin R. and Elia E. 2002. Discussion of metrics for distributed project management: Preliminary findings. In proceedings of the 35th annual Hawaii international conference on system sciences HICSS'02. IEEE. 10 p.

Buse R. P. L. and Zimmermann T. 2010. Analytics for software development. In proceedings of the FSE/SDP workshop on future of software engineering research. ACM. Pp. 77–80.

Card D. 2003. Integrating practical software measurement and the balanced scoreboard. In proceedings of the 27th annual international COMPSAC 2003. Pp. 362–363.

Carmel E. 1999. Global software teams: Collaborating across borders and time zones. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Carmel E. and Tija P. 2005. Offshoring information technology: Sourcing and outsourcing to a global workforce. Cambridge, the United Kingdom: Cambridge University Press.

Casey V. and Richardson I. 2008. Virtual teams: Understanding the impact of fear. Software Process Improvement and Practice 13(6): 511–526.

CHAOS Report. 2014. The Standish Group report: Project Smart. Available from: http://www.projectsmart.co.uk/docs/chaos-report.pdf. Accessed 15.10.2014.

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

Coman I. D., Sillitti A. and Succi G. 2009. A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In proceedings of the 31st international conference on software engineering, ICSE 2009; May 16–24. IEEE. Pp. 89–99.

Creswell J. W. 2009. Research design: Qualitative, quantitative, and mixed methods approaches (third edition). Thousand Oaks, CA: SAGE Publications, Inc.

da Silva F. Q. B., Costa C., França A. C. C. and Prikladinicki R. 2010. Challenges and solutions in distributed software development project management: A systematic literature review. In proceedings of international conference on global software engineering (ICGSE2010). IEEE. Pp. 87–96.

Damian D. E. and Zowghi D. 2003. An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In proceedings of the 36th annual Hawaii international conference on system sciences (HICSS'03). 10 p.

Dullemond K. and van Gameren B. 2013. What distributed software teams need to know and when: An empirical study. In proceedings of the 8th international conference on global software engineering (ICGSE 2013). IEEE. Pp. 61–70.

Easterbrook S., Singer J., Storey M. and Damian D. 2008. Selecting empirical methods for software engineering research. In: Shull, F. and Singer, J., (eds.). Guide to advanced empirical software engineering. Springer. Pp. 285–311.

Eskeli J., Maurolagoitia J. and Polcaro C. 2011. PSW: A framework-based tool integration solution for global collaborative software development. In proceedings of the sixth international conference on software engineering advances (ICSEA'11). Barcelona, Spain. Pp. 124–129.

Eskeli J. and Maurolagoitia J. 2011. Global software development: Current challenges and solutions. In proceedings of the 6th international conference on software and data technologies, ICSOFT 2011. Pp. 29–34.

Fenton N. E. and Pfleeger S. L. 1998. Software metrics: A rigorous and practical approach. 2nd ed. Boston, MA: PWS Publishing Co.

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

Fryer K. and Gothe M. 2008. Global software development and delivery: Trends and challenges. Retrieved June 7, 2009, from http://www.ibm.com/developerworks/rational/library/edge/08/jan08/fryer_gothe/index.html. Accessed 15.10.2014.

Herbsleb J. and Mockus A. 2003. An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering 29(6): 481–494.

Herbsleb J. D. 2007. Global software engineering: The future of socio-technical coordination. In proceedings of future of software engineering FOSE '07. IEEE Computer Society. Pp. 188–198.

Herbsleb J. D. and Moitra D. 2001. Global software development. IEEE Software 18(2): pp. 16–20.

Herbsleb J. D., Mockus A., Finholt T. A. and Grinter R. E. 2000. Distance, dependencies, and delay in a global collaboration. In proceedings of the ACM conference on computer supported cooperative work. ACM. Pp. 319–328.

Herbsleb J. D., Paulish D. J. and Bass M. 2005. Global software development at Siemens: Experience from nine projects. In proceedings of the 27th international conference on software engineering (ICSE 2005). IEEE. Pp. 524–533.

Hevner A. and Chatterjee S. 2010. Design research in information systems: Theory and practice. Springer.

Hofstede G. 2001. Culture's consequences. Comparing values, behaviors, institutions, and organizations. 2nd edition. London: Across Nations. Sage Publications.

Holmstrom H., Conchuir E. O., Ågerfalk P. J. and 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). IEEE. Pp. 3–11.

Hyysalo J., Parviainen P. and Tihinen M. 2006. Collaborative embedded systems development: Survey of state of the practice. In proceedings of the 13th annual IEEE international symposium and workshop on engineering of computer based systems (ECBS 2006). IEEE. Pp. 1–9.

IEEE Std 1061-1992. 1992. IEEE standard for a software quality metrics methodology. Piscataway, NJ: IEEE Computer Society.

Järvinen P. 2012. On research methods. Tampere, Finland: Tampereen Yliopistopaino Oy.

Jiménez M., Piattini M. and Vizcaíno A. 2009. Challenges and improvements in distributed software development: A systematic review. Advances in Software Engineering Jan-2009 (No. 3): 1–16.

Jones C. 2004. Software project management practices: Failure versus success. CrossTalk: The Journal of Defense Software Engineering, October 2004. 17 p.

Kaplan R. S. and Norton D. P. 1992. The balanced scorecard-measures that drive performance. Harward Business Review (No. 92105): 71–79.

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); 1.–3. July; San Francisco, USA. Pp. 489–496.

Komi-Sirviö S., Parviainen P. and Ronkainen J. 2001. Measurement automation: Methodological background and practical solutions a multiple case study. In proceedings of the 7th international software metrics symposium, METRICS 2001. IEEE. Pp. 306–316.

Korhonen K. and Salo O. 2008. Exploring quality metrics to support defect management process in a multi-site organization – A case study. In proceedings of 19th international symposium on software reliability engineering (ISSRE). IEEE. Pp. 213–218.

Lavazza L., Morasca S., Taibi D. and Tosi D. 2012. On the definition of dynamic software measures. In proceedings of the ACM-IEEE international symposium on empirical software engineering and measurement. ACM. Pp. 39–48.

Lawler J. and Kitchenham B. 2003. Measurement modeling technology. IEEE Software 20(3): 68–75.

Lawrie G. and Cobbold I. 2004. Third-generation balanced scorecard: Evolution of an effective strategic control tool. International Journal of Productivity and Performance Management 53(7): 611–623.

Lewis R. D. 2006 (revised edition). When cultures collide: Managing successfully across cultures. Boston, MA & London, UK: Nicholas Brealey Publishing.

Lewis R. D. 1999. Cross cultural communication: A visual approach. Transcreen Publications.

Lings B., Lundell B., Agerfalk P. J. and Fitzgerald B. 2007. A reference model for successful distributed development of software systems. In proceedings of the 2nd IEEE international conference on global software engineering, (ICGSE 2007). IEEE. Pp. 130–139.

Lotlikar R. M., Polavarapu R., Sharma S. and Srivastava B. 2008. Towards effective project management across multiple projects with distributed performing centers. In proceedings of IEEE international conference on services computing (CSC'08). IEEE. Pp. 33–40.

Mingguang Z., Haohua Z., Weiyi Q., Shijun M. and Chuanyi W. 2009. The measurement and evaluation for large-scale object-oriented software system. In proceedings of the 9th international conference on Hybrid intelligent systems, HIS'09. IEEE. Pp. 70–73.

Misra S. 2009. A metric for global software development environment. In proceedings of the Indian national science academy. Pp. 145–158.

Noble D. 2004. Knowledge foundations of effective collaboration. In proceedings of the 9th International Command and Control Research and Technology Symposium, September 14-16, Copenhagen, Denmark.

Noll J., Beecham S. and Richardson I. 2010. Global software development and collaboration: Barriers and solutions. ACM Inroads 1(3): 66–78.

Paasivaara M. and Lassenius C. 2003. Collaboration practices in global inter-organizational software development projects. Software Process: Improvement and Practice 8(4): 183–199.

Parviainen P. 2012. Global software engineering. Challenges and solutions framework. Doctoral Dissertation, VTT Science 6. Espoo, Finland: VTT. 106 p. + app. 150 p.

Parviainen P., Eskeli J., Kynkäänniemi T. and Tihinen M. 2008. Merlin collaboration handbook – challenges and solutions in global collaborative product development. In proceedings of the 3rd international conference on software and data technologies; July 5–8; Porto, Portugal. Pp. 339–346.

Peixoto C. E. L., Audy J. L. N. and Prikladnicki R. 2010. Effort estimation in global software development projects: Preliminary results from a survey. In proceedings of international conference on global software engineering. IEEE Computer Society. Pp. 123–127.

Pesola J.-P., Eskeli J., Parviainen P., Kommeren R. and Gramza M. 2008. Experiences of tool integration: Development and validation. In proceedings of international conference on interoperability of enterprise, software and applications. enterprise interoperability III – new challenges and industrial approaches. Berlin, Germany: Springer. Pp. 499–510.

Ruhe G. 2003. Software engineering decision support – a new paradigm for learning software organizations. In: S. Henninger and F. Maurer (eds.) Advances in learning software organizations, LSO 2003. Berlin Heidelberg: Springer. Pp. 104–113.

Sangwan R., Bass M., Mullick N., Paulish D. J. and Kazmeier J. 2006. Global software development handbook. CRC Press.

Sengupta B., Chandra S. and Sinha V. 2006. A research agenda for distributed software development. In proceedings of the 28th international conference on software engineering. ACM. Pp. 731–740.

Shewhart W. A. 1939. Statistical method from the viewpoint of quality control. Washington: Graduate School of Agriculture.

Simmons D. B. and Ma N. K. 2006. Software engineering expert system for global development. In proceedings of 18th IEEE international conference on tools with artificial intelligence (ICTAI'06). IEEE. Pp. 33–38.

Soubra H., Abran A., Stern S. and Ramdan-Cherif A. 2011. Design of a functional size measurement procedure for real-time embedded software requirements expressed using the simulink model. Joint conference of the 21st international workshop on software measurement and the 6th international conference on software process and product measurement. IEEE. Pp. 76–85.

Tihinen M. 2001. Analysis of the quality measurement processes in software production. Secondary Subject Thesis. Oulu, Finland: University of Oulu, Department of Information Science.

Umarji M. and Shull F. 2009. Measuring developers: Aligning perspectives and other best practices. IEEE Software 26(6): 92–94.

van Solingen R. and Berghout E. 1999. The goal/question/metric method: A practical guide for quality improvement of software development. McGraw-Hill.

Vierimaa M., Ronkainen J., Salo O., Sandelin T., Tihinen M., Freimut B. and Parviainen P. 2001. Comprehensive collection and utilisation of software measurement data. VTT Publications 445. Espoo, Finland: VTT.

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

Welborn R. and Kasten V. 2003. The Jericho principle: How companies use strategic collaboration to find new sources of value. Hoboken, NJ: John Wiley & Sons.

Wohlin C., Höst M. and Henningsson K. 2003. Empirical research methods in software engineering. In: R. Conradi and A. I. Wang (eds.) Empirical methods and studies in software engineering. Springer. Pp. 7–23.

Wu C., Chang W. and Sethi I. K. 2009. A metric-based multi-agent system for software project management. In proceedings of the 8th IEEE/ACIS international conference on computer and information science (ICIS 2009). IEEE. Pp. 3–8.

Yin R. K. 2009. Case study research: Design and methods. 4th ed. Los Angeles: SAGE Publications.

Zhang H., Zhao H., Cai W., Zhao M. and Luo G. 2008. A metrics suite for static structure of large-scale software based on complex networks. In proceedings of international conference on intelligent information hiding and multimedia signal processing, IIHMSP'08. IEEE. Pp. 512–515.

PAPER I

Lessons learned by participants of distributed software development

In: Knowledge and Process Management,
Vol. 2, No. 2, pp. 108–122.

Knowledge and Process Management Volume 12 Number 2 pp 108–122 (2005)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/kpm.225

■ *Research Article*

Lessons Learned by Participants of Distributed Software Development

Seija Komi-Sirviö* and Maarit Tihinen

Technical Research Centre of Finland, VTT Electronics, Oulu, Finland

The maturation of the technical infrastructure has enabled the emergence and growth of distributed software development. This has created tempting opportunities for companies to distribute their software development, for example, to economically favourable countries so as to gain needed expertise or to get closer to customers. Nonetheless, such distribution potentially creates problems that need to be understood and addressed in order to make possible the gains offered. To clarify and understand the most difficult problems and their nature, a survey of individuals engaged in distributed software development was conducted. The purpose of this survey was to gather and share lessons learned in order to better understand the nature of the software development process when operating in a distributed software development environment and the problems that may be associated with such distributed processes. Through a clear appreciation of the risks associated with distributed development it becomes possible to develop approaches for the mitigation of these risks. This paper presents the results of the survey, focusing on the most serious problems raised by the respondents. Some practical guidelines that have been developed by industry to overcome these problems are also briefly summarized. Copyright © 2005 John Wiley & Sons, Ltd.

INTRODUCTION

Distributed software development enables software production to take place independently of the geographical location of the individuals/organizations concerned. Software subcontracting, partnership-based development and global business ventures are all different business strategies exploiting the advantages that such distributed is expected to bring. Unfortunately, distributed software development projects have inherited the same problems that single-site software projects have been struggling with. Thus, distributed projects suffer equally from quality, schedule and cost related problems—the distribution only makes these harder to handle. In addition, distribution itself may create time slippage problems. A recent study shows that the physical distance between development sites alone is likely to create delays in work (Herbsleb *et al.*, 2001). Problems in task coordination, project management and communication have also been reported (Hersleb and Moitra, 2001). Over and above this, distribution has also introduced new specific problems (De Souza *et al.*, 2002).

Despite the challenges entailed in distribution, distributed software development is a development strategy that is in increasing favour with the industry (Herbsleb *et al.*, 2001; Battin *et al.*, 2001; Ebert and De Neve, 2001). The expected benefits, such as the possibility for high-speed development through the use of individuals/teams in different time zones (Hersleb and Moitra, 2001; Gorton and Motwani, 1996; Mockus and Herbsleb, 2001), the employment of more skilful staff, the lower development costs (Hersleb and Moitra, 2001; Press, 1993) and the ability to respond to local customers' needs, are expected to outweigh the risks involved.

*Correspondence to: Seija Komi-Sirviö, Technical Research Centre of Finland, VTT Electronics, P.O. Box 1100, FIN-90571 Oulu, Finland. Email: Seija.Komi-Sirvio@vtt.fi

In prior research a number of unique topics have been studied by researchers that relate to distributed software engineering, for example, requirements engineering (Lloyd *et al*., 2002), project management (Lam and Maheshwari, 2001; Gaeta and Pierluigi, 2002; Aversano *et al*., 2003) and configuration management (Van der Hoeak *et al*., 1996). However, the viewpoints of these studies are biased towards the use of software development tools. For example, solutions are offered by different researchers in the form of a software configuration management tool (Surjaputra and Maheswari, 1999), a process support system (Gianpalo and Ghezzi, 1999) or a virtual corporation negotiation approach (Kötting and Maurer, 1999). More recently, some interesting industrial experiences of distributed software development have been published (e.g. Herbsleb *et al*., 2001; Battin *et al*., 2001; Ebert and De Neve, 2001; Karlsson *et al*., 2000). In all, however, a thorough understanding of the complex of problems connected with distributed software development has not been developed.

The aim of this research is to gain a better understanding of the problems faced when software development projects are distributed over multiple sites in different geographical locations. The purpose of the survey was to enable ranking of problem areas according to their frequency of occurrence and to gather practical knowledge and examples of the problems experienced along with the potential solutions that have been developed and tested by developers. Being more aware of possible pitfalls and potential risks, those involved with the development of distributed software projects should therefore be in a better position to successfully plan and execute these projects.

As we have noted above, there may be several reasons for organizations to distribute their software development. One of the potential reasons is the possibility of using the fact that individuals/groups reside in different time zones to enable rapid development, for example distributing development and test sites across different time zones, and then synchronizing them to a continuous 24-hour product development and testing cycle (Gorton and Motwani, 1996). However, this hypothesis was not verified by the results of this survey as there were no indications that this was either attempted or even desired. Primarily, the motivation for distribution was '*peopleware*' related: the needed knowledge was distributed, there was no local expertise to solve a development problem, or the local demand for software development was insufficient. The survey also sought to identify factors that would support project distribution. It turned out that the duration and total effort associated with the project were not as significant as the size of the project measured in terms of the number of people involved.

This paper is organized as follows. In the next section the background information of the survey is presented. In the third section the results of the survey are introduced and the improvement approaches raised are analysed. In addition, based on those results and issues, observations of distributed software development process will be discussed. The final section presents the conclusions of the survey.

SURVEY BACKGROUND

Knowledge acquisition was carried out in the form of a questionnaire. The semi-structured questionnaire, containing a large set of open and closed questions relating to distributed software development, was sent by mail and e-mail to the recipients (it was also accessible via the Internet). The questionnaire covered the following topic areas:

- characterization of the organization;
- characterization of the distributed projects;
- utilization rate of various communication tools;
- problems and the solutions developed to overcome them;
- advantages of distribution; and
- overall satisfaction.

The survey was conducted during the summer of 2002. The questionnaire was posted to 44 organizations in Finland and it was also e-mailed to over 200 organizations around the world. The total number of responses was 31, representing 21 different organizations. The regular mail proved to be the best way of reaching the respondents: out of retrieved 31 replies 24 were received by post. Conversely, e-mail turned out to be a very inefficient way to reach respondents; only seven replies were retrieved using e-mail. The replies came mainly from Finland, but some also from the Netherlands and the USA. Four replies were eliminated from the analysis due to the fact that they were received from organizations not carrying out distributed software development. It turned out that these organizations were distributed globally. Thus, in the final analysis there were 27 responses included in the survey.

In this paper, we concentrate on analyzing problems relating to distributed software development. Most of the responses to the survey came from the telecommunication or wireless telecommunication industries (see Figure 1).

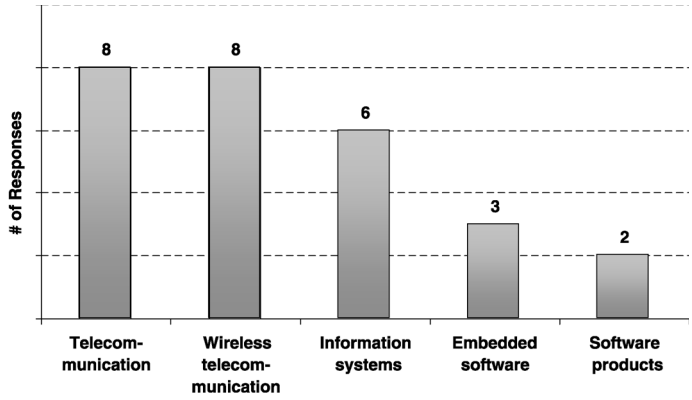

Figure 1 Business scope of organizations

Within the survey sample software development distribution was very extensive and also global. In only four cases (14.8%) did respondents report involvement in software development distribution within one country. Ten responses (37.0%) referred to software projects which were distributed within one continent, and 13 responses (48.1%) reported that software development projects had been undertaken which were distributed over two or several continents. Figure 2 illustrates the positions of those who responded to the survey. Analysing these positions we can conclude that to a large extent they are characterized by extensive experience in software development.

Table 1 separates the responses by viewpoint (organizational or project viewpoint) and furthermore by the extent to which distribution was implemented. The survey was carried out within the Finnish research programme; thus the responses are mainly from Finland. The number of software development projects involving teamwork between continents—namely Europe and the USA and Europe and Asia (13 responses, 48.1%)—was essentially equal to the number of projects involving cooperation within Europe (14 responses, 51.9%).

Figure 3 shows a geographical distribution of software development represented by the survey respondents. The arrows in Figure 3 illustrate the direction (to a country or a continent) in which a respondent reported that their software development had been distributed. If the respondent had named a city, a bullet is used as an identifier of that place. Each line starts from a city corresponding to the home address of the respondent. Furthermore, some lines have been drawn in bold, indicating various responses reporting distribution to the same city, country or continent. Two anonymous responses have not been included in the figure since their home office is not known. One

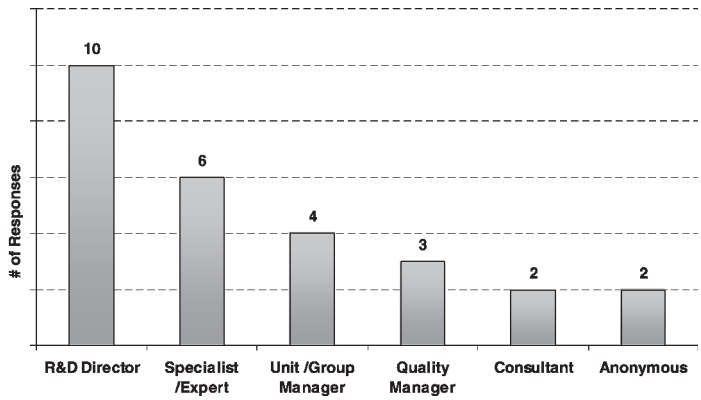

Figure 2 Positions of respondents

Table 1 Characteristics of responses

Response viewpoint	Extent of software development distribution			
	Within one country	Within one continent	Between two continents	Between three or more continents
Organization	2	5	5	3
Project	2	5	3	2
Total number of answers	4	10	8	5

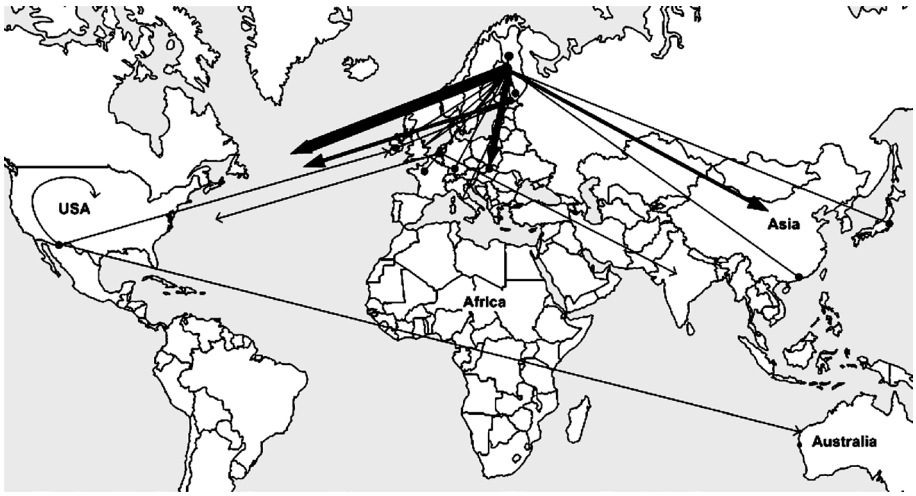

Figure 3 Geographical distribution of projects

anonymous answer reported that their software development had been distributed between two continents, and another reported development being distributed between three continents.

SURVEY RESULTS

In this section the results of the survey are presented. First, the characteristics of the organizations and projects of those taking part in the survey are outlined. Second, the problems introduced by the distribution of software development are discussed along with the solutions to overcome these problems as described by respondents. Lastly, based on analysis of the preceding data, potential success factors are identified.

Characteristics of the organizations and the projects

Our survey results indicate that software development distribution is carried out not only by large organizations but also by small and medium-sized enterprises: 47% of the respondents were employed by companies having fewer than 500 employees in total. Figure 4 depicts the distribution of responses by organization size (measured in terms of the total number of employees).

Most companies taking part in the survey were operating in the area of embedded software. Commonly, the distributed development concerned a proprietary product of a company, but there were

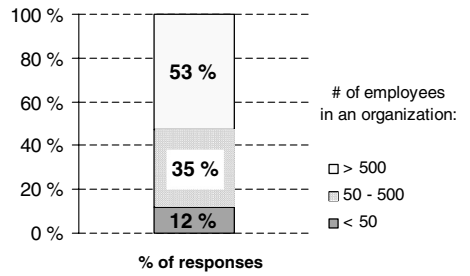

Figure 4 Distribution of responses by organization size

also some responses that reported on companies developing software components for clients. The business governance strategies adopted in distribution may take various forms, such as partnering, subcontracting or the use of sister sites. The extent of distribution, denoted by the distance between cooperating development offices, vary from those distributed over a single town to those distributed between several continents. However, it is appropriate to observe that the results presented in this paper primarily concern issues typical of organizations distributing their software development activities both extensively and globally. Over half of the responses (52%) had distributed development activities to different continents, and more than two-thirds of them (74%) to different countries. While subcontractors were used extensively, the extent of distribution in this case was slightly smaller: less than one-third (30%) had subcontractors in different continents and 41% had them in different countries. In all, a total of 85% of the respondents had used subcontracting as an approach to software development.

The characteristics of the distributed projects were determined in terms of duration, project size in person years and the size of the development team. It turned out that the duration of a distributed project was rarely longer than 2 years; 78% of the projects were shorter than that and, of these, one-third were of less than 1 year. Furthermore, the size of the project measured in person years turned out to be either rather large—almost half of the projects were over 20 person years in size—or rather small, half of the projects being under 10 person years in size. Only 11% of the projects studied fell in the category between 10 and 20 years. The size of the development team showed the least variation: in most cases (56%) the team was smaller than 20 software developers. The results suggest that globally distributed projects tend to be rather large measured in terms of person years, while the development team was often kept quite small.

A closer look at the number of software developers in the organizations represented by the respondents reveals that the majority of answers (67%) were provided by respondents from companies having more than 100 software developers, and one-third (33%) of the responses were received from individuals employed by organizations with under 100 software developers. These two groups of organization were studied to detect possible differences in their characteristics, and a noteworthy difference was identified concerning the size of distributed projects in person years (see Figure 5).

Figure 5 shows that when the number of software developers in an organization is under 100 the size of distributed projects is most often (67%) smaller than 10 person years.

Problems in distributed software development

In the survey, respondents were given the choice of identifying eight different problem areas. These problem areas were identified on the basis of the descriptions found in the literature (Herbsleb *et al.*, 2001; Mockus and Herbsleb, 2001; Karlsson *et al.*, 2000; Niederman *et al.*, 1993; Zoran *et al.*, 1995). Respondents were asked to check all the problems they had experienced in their projects and to describe each of the problems identified in more detail. Respondents were also provided with an

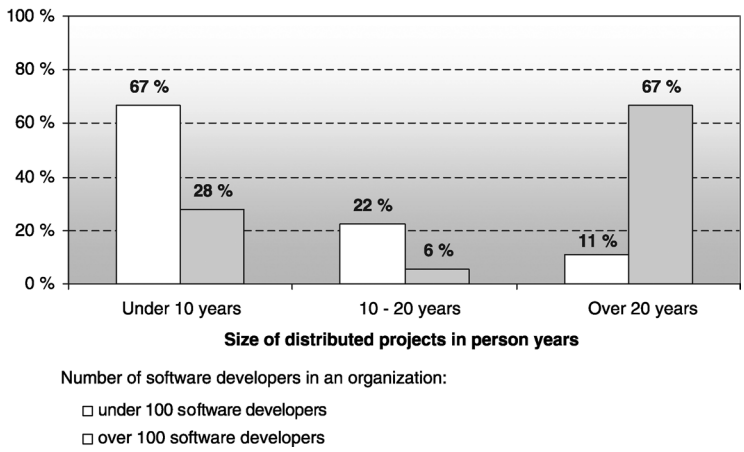

Figure 5 Size of distributed projects and number of software developers

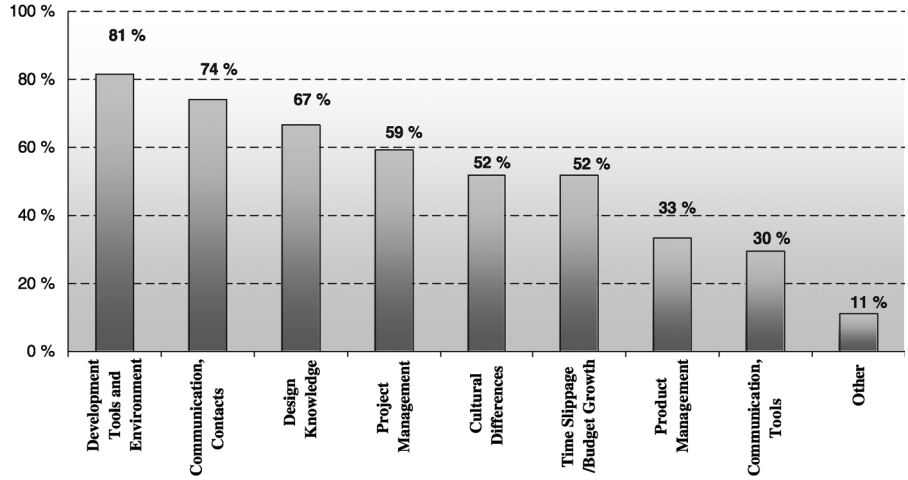

Figure 6 Problem areas

opportunity to identify problems that were not listed. This '*other*' category was used by only two respondents, which gives some confidence that the proposed classification of problems adequately covered the key issues associated with the projects identified by the respondents. In addition to problem descriptions, respondents were also asked to describe the solutions they had developed to overcome the problems encountered and to evaluate the success of these solutions. The overall distribution of answers among problem areas is presented in Figure 6.

Development tools and environment

While the existing technical infrastructure seems to provide adequate support for distributed software development, respondents indicated that the development environment is not yet mature. Projects are most often undermined by poor software development environments and tools. Thus, tool-related issues were most commonly experienced as a problem area, with a selection rate of 81%. The problems identified were diverse, though two main problem streams could be clearly extracted, namely those concerning network connections and the compatibility of the tools used.

In the case of issues relating to network connections the specific issues that were identified related to the reliability and usability of the network being unsatisfactory. For example, the network connection was deemed to be too slow, with its reliability further impaired by occasional server failures. Connections that were too slow were considered to be not only a source of frustration due to increased waiting time, but were also reported to cause usability problems with development tools. One respondent described the problem as follows: 'development tools are based on the assumption that the network is extremely fast'.

The solutions that were adopted to address network problems were varied; one solution was to increase the bandwidth between the sites, which, however, failed to have the desired effect on speed. In this case the problem may not have been the available bandwidth but rather latency in response times. In several projects the slowness and unreliability of the network were successfully reduced to an acceptable minimum level by changing the development strategy from a synchronous approach to an asynchronous one. That is, instead of having one central database for the project, each development site had its own local database, and replication and reconciliation were done once a day. This time-synchronized resolution may be a good solution if real-time development is not obligatory. Whether a time-synchronized solution is favoured or not, the distributed development environment emphasizes the need for having proper configuration management tools and processes in place.

Establishing a uniform software development environment is a challenging task. Not only were a diverse set of tools reported to cause problems but also the different versions of a single tool potentially caused problems as well. As it turned out respondents are relatively well aware of the criticality of establishing a compatible environment; the problem is how such an environment can be established successfully. Respondents indicated

that their readiness to change development tools was quite low; development sites were reported to be reluctant to change the development tools they were already familiar with. In one company it was considered a highly sensitive question which site had the dominant role in defining which tools were to be used. Forcing the sites to use identical tools appeared to make the problem even worse. The respondents did not have any easy or assured solutions to offer to solve this issue. Some respondents described how they had had to visit the sites themselves to be able to clarify what tools were used, what they were capable of and how the interoperability with the various tools and their different versions could be achieved without changing the existing tool configuration.

The respondents also thought about the reasons for the problems concerning development tools and the development environment. The obvious technology-based problem appeared to be aggravated by distance and various human factors. Cultural differences between countries and organizations were pointed out as a potential problem factor, along with the lack of communication and missing face-to-face-meetings; these are likely to cause information and knowledge deficiencies, further provoked by the different time zones. Configuration and document management tools, test environment, and replication and synchronization of artifacts are critical for distributed software development and therefore need to be carefully studied and planned case by case. Obviously, the most effective stage to address these tool-related issues is when the distributed software development project is set up.

In summary, the solutions disclosed by respondents to overcome slow and unreliable network were:

- changing the development strategy from synchronous to asynchronous and replicating once a day; and
- increasing the bandwidth between the sites to improve speed.

In addition to the solutions above, the respondents gave the following practical advice for controlling the tool environment:

- define and document acceptable tools and versions for the whole project life cycle;
- define configuration and version management tools and practices;
- get official, explicit approval for the plan from all parties involved; and
- arrange for the main developer site to take the lead and responsibility for the tool environment and for organizing identical tools for all sites.

Communication and contacts

The area 'communication and contacts' was selected as a source of problems for distributed projects by the majority (74%) of the respondents. The result itself was not a great surprise but the reasons stated by respondents for this selection were interesting. There were two factors repeatedly mentioned as causing problems with respect to communications and contacts: cultural differences (including language skills) and physical distance.

Cultural differences may give rise to misunderstandings and exacerbate problems in oral communication. The respondents described that sometimes the level of language skills alone was often so low that fluent conversation became impossible, most particularly telephone conferences. The problems that cultural differences are likely to generate are introduced in more detail later in this article.

The survey clarified how the frequency of face-to-face meetings changed with increasing distance. It turned out that when the development was done at a single site, but in different buildings, 88% of the projects involved having face-to-face meetings one to three times per week or even every day. When there were several sites involved either in a single city or in different cities, the frequency of the face-to-face meetings dropped, with only 4% having weekly meetings. When the sites were situated in different countries, the most active projects in terms of communication still had face-to-face meetings, one to three times per a month (48%), but the rest of the projects had then only one to three times per year (see Figure 7). This result supports the respondents' view that physical distance causes miscommunication and that face-to-face meetings really are the best means of communication.

Time zones play a role in globally distributed projects. Especially when development is distributed over continents with extensive time differences, this structure creates challenges and causes practical problems for projects. The survey results verify that increase in physical distance increases problems as well (Herbsleb *et al.*, 2001; Battin *et al.*, 2001). The time difference being the effective working days in different time zones reduces both the willingness—and opportunities—to have meetings. The reduction in the number of project meetings, along with language barriers and cultural differences (county or company related), causes a number of problematic situations. Lack of knowledge and misunderstandings are reported by respondents to have led to redundant work or no work at all due to mistaken assumptions concerning who is in charge of different stages in the project

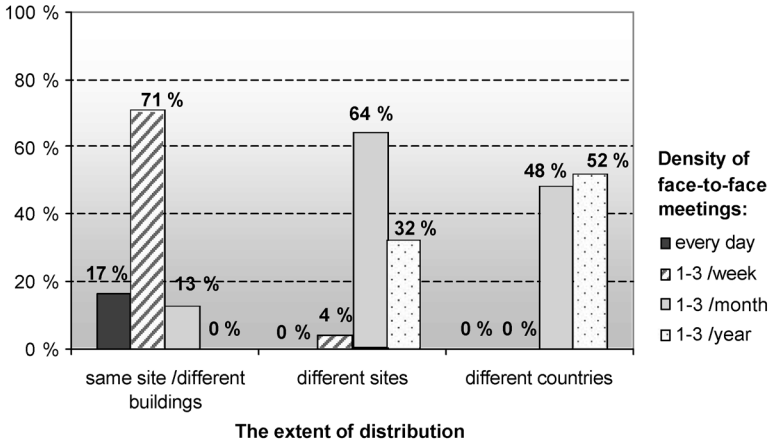

Figure 7 Frequency of face-to-face meetings

being developed. Respondents also mentioned that distance is likely to makes it easier to hide possible problems and to withdraw from decision making.

Respondents use various strategies intended to substitute for the missing face-to-face communication. The most common solutions applied are tele- or videoconferences and e-mail. However, even videoconference is regarded as problematic due to the connection difficulties that are often experienced. Despite this, communication problems were not very often linked to the tools used; only 30% of respondents regarded communication tools as a problem (see Figure 6). It may thus be speculated that if communication tools are experienced as good enough and communication is still a problem, better communication tools are not likely to solve the underlying communication problem.

The foundations for effective communication are laid at the beginning of the project. Informal team-building sessions are identified by respondents as one of the main means of building trust and feelings of togetherness. Becoming thus acquainted with each other, face-to-face substitutes, such as e-mail, NetMeeting and tele- and videoconferences, become ways of improving communication. Some respondents suggest that project members should know each other so well that the barrier to contact becomes non-existent.

Another common and recommended means of minimizing communication-related problems is to decrease communication needs and contact points to a minimum by splitting the project into smaller, independent units managed by a local manager. If no local project manager can be appointed, at least a contact person should be named for answering questions and acting as a contact point.

In summary, the following strategies are considered successful in improving communication in practice:

- informal team-building sessions and face-to-face meetings, especially at the beginning of the project;
- decreasing the need for contacting other team members by splitting projects into smaller, independent and more manageable units; and
- appointing a contact person from each site.

Design knowledge

Design knowledge was the third most frequently selected problem area by all respondents (67%). The problem descriptions relating to this issue covered the following design mediating-related issues: *'Interpretation of specification, understanding of design rationale'* and *'Difficult to transfer knowledge'*. For example, if architecture design is done at a site different from where the implementation takes place, efforts must be made to ensure that the design rationales are understood and communicated across the sites, so as to verify that they are understood correctly. Only a few respondents report problems originating from partner site incompetence or insufficient ability to carry out their development tasks. The problem descriptions indicate mainly problems relating to knowledge transfer and communication. They also noted that knowledge is typically distributed asymmetrically.

Knowledge transfer and sharing was recognized as a bottleneck especially when the chief architect and software engineers were working at different sites, as was often the case. Here again, the role of

face-to-face meetings was highlighted and regarded as the fastest way to solve a design problem or to obtain an answer to a specific question. Knowledge transfer solely via design documents was regarded as a slow and laborious process. This observation addresses the need for having a design document with clear and adequate structure, content and level of detail to answer the requirements of the distributed software development.

The strategies used to overcome design knowledge-related problems do not differ significantly from the solutions already discussed. Once again, face-to-face meetings are regarded as essential to build a common understanding of terminology, approaches and the application area. Kick-off meetings at the beginning of the project and technical meetings in the design phase are highly recommended. To attenuate the communication needs, it is recommended that the project be split into smaller parts and that the responsibility for the parts is distributed among sites. Moreover, practical guidelines for using common databases, tools, and version and configuration management were listed as helpful. In addition, training material available for all in an electronic format made it easier for engineers to acquire and use available knowledge.

As a summary, the following factors are considered to be important for improving a design phase:

- face-to-face kick-off and technical meetings to discuss design rationale, terminology and application area issues;
- division of work and responsibility into smaller units;
- practical guidelines for developing design documents and using development tools; and
- training material provided in electronic form.

Project management

Project management is identified as a source of problems by 59% of respondents, making it the fourth biggest problem area. Project management challenges are harder to solve in a distributed environment than in a centralized development environment. It is noted that in 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. Furthermore, if this need is not recognized at the beginning of the project, uncertainty, misunderstandings and management problems are most likely to appear later. The manager of a globally distributed project has to have varied abilities and knowledge, such as cultural knowledge and communication skills in addition to technical competence and particularly good project management capabilities. One respondent gave the following example: 'In centralized project management, a project manager may be far away from development groups, which creates a visibility problem, and makes it easier to hide possible problems.' In other words distribution, for example, makes time slippage more difficult to estimate and control due to the decreased visibility. Time slippage and budget growth are listed as a problem area in 52% of all responses, which is, naturally, an issue relating to project management in general. Generally speaking the budget can be adjusted to take into account the higher costs involved due to distribution, for example additional planning activities, communication, and lead to extra travelling and meeting costs. However, if the project manager does not know the software engineers involved in the development at the other site(s), it becomes difficult for him or her to assess whether given estimates are realistic or not. In addition to unrealistic estimates, the project manager may suffer from problem hiding, which is more likely to take place in a distributed environment for a longer period of time than in single-site development. Eventually, this may cause serious problems in keeping to the schedule and achieving project development goals. One respondent captures his experience in the following down-to-earth heuristic: 'The farther the subcontractor the bigger the delays in the schedule.'

Several other problems are mentioned as well, such as failure to inform other sites of decisions or changes that have been made. Difficulties in getting the information as requested were also brought up, e.g. delays in getting status reports. Again, the issues refer mainly to knowledge management and communication-related problems.

Respondents have good experience of solid project management practices with strict control as solutions to the types of project management problems discussed above. Breaking down project tasks into weekly delivery results is reported to be an efficient way to track the progress of development projects. Another proven strategy is to plan development blocks so that they can be independently developed by different sites. If this solution is adopted, a local project manager can take over some planning and follow-up activities from the main project manager. In addition, clearly established rules, definitions of responsibilities, results and timetables along with regular meetings and the management and control of process are reported to be highly important in a distributed software development environment.

In summary, the following actions have helped in the management of distributed software development:

Table 2 Extent of software development distribution and reported problems

Selected problem areas	Extent of software development distribution			
	Within one country[a] (4 replies in total)	Within one continent (10 replies in total)	Between two continents (8 replies in total)	Between three or more continents (5 replies in total)
Communication—contacts	3	7	5	4
Communication—tool	1	2	4	1
Cultural differences	1	3	7	4

[a]Although four replied that their project distribution was carried out within one country, two of them had subcontractors in a different country or countries.

- detailed up-front planning and strict control activities; and
- splitting the project into sub-projects with local project managers.

Cultural differences

Cultural differences are mentioned as a problematic issue in 52% of all responses. Table 2 depicts how cultural and also communication problems grow while software project distribution extends over one or several continents.

Different and divergent values and perceptions may cause misunderstandings and even dissatisfaction within the project. Depending on the cultural differences, be they related to company culture or the culture of a country, the same action can have different interpretations. Cultural differences create possibilities for misinterpretation, especially when these cultural differences are not appreciated, let alone understood. One respondent gave an example of different viewpoints to problem reporting: at one site it was a regular action originating from a basic development procedure; at another site such reporting was understood as an insult. It is also important to pay attention to how things are expressed; when communicating in a foreign language, it is easy to send unintentional messages. In addition, terms and concepts may vary between different countries and even within one country. Commitment to decisions and timetable is also claimed to vary between countries and continents.

To alleviate cultural differences, and to reconcile conceptions, approaches and terminology in use, it is recommended that actions be taken to enhance communication inside the project. Face-to-face communication appears to be an effective means of lowering thresholds caused by cultural differences. Training and common sense are also considered to have an important role in tackling the cultural differences in distributed projects. As noted above, kick-off meetings at the beginning of the project and technical meetings during the design phase are highly recommended.

To summarize, the following factors are reported to be important for improving the culture-related problems associated with distributed development projects:

- enhanced communication throughout the project in order to build understanding between the sites;
- defining and using predefined terminology; and
- improving language skills and developing and sharing knowledge concerning cultural issues and customs.

Requirements engineering: the main software error source

Requirements engineering-related issues were stressed when respondents were asked to determine the main sources of software error. The predefined alternatives included various error sources ranging from design, coding, interfaces, environment problems and security vulnerabilities to requirements engineering issues. We anticipated requirements engineering to be one of the principal problems (Damian, 2002) and therefore we divided it into three categories. There was also one open choice (Other, describe) in the questionnaire, which was, however, not selected by anyone. The most significant problem source was marked '1' and the least significant '8'.

Figure 8 illustrates how requirements engineering is ranked for software errors. 'Misinterpretation of requirements' was ranked to be the most significant error source, with 74% of answers ranking it as 1, 2 or 3. Problems also arise from insufficient communication, poor quality of requirement documents and the requirements engineering process

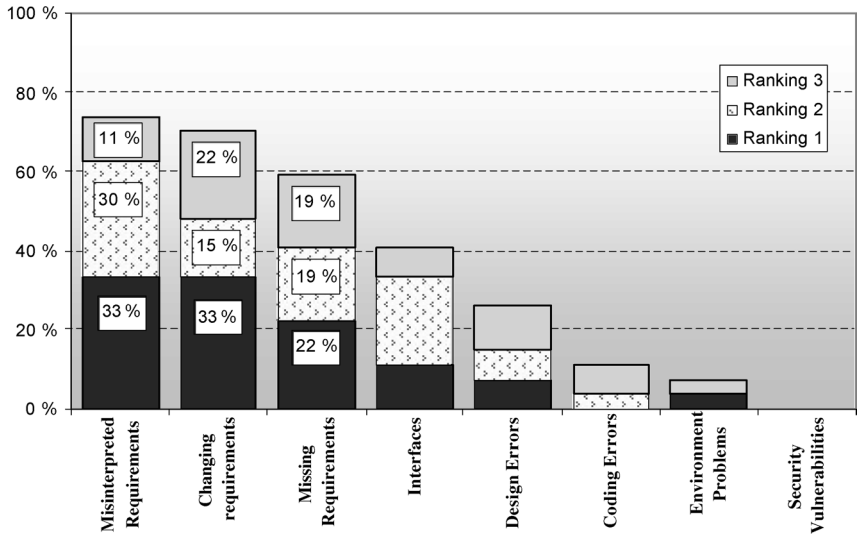

Figure 8 Major sources of software errors

itself. Detailed documentation can reduce misunderstanding, but it is also considered a slow and cumbersome way of transferring information. 'Changing requirements' was ranked as the second most significant error source by 70% of the respondents. While changes in requirements are likely to lead to errors in software development in general, in distributed software development these changes are even more difficult to manage and communicate due to, for example, the number of stakeholders and the distance involved. Problems originating from 'missing requirements', which was ranked third (59%), can be successfully prevented by careful planning and execution of the requirements engineering process.

The importance of organization size

When further analysing the survey results, differences were detected between large organizations (over 500 employees) and small and medium-size organizations (under 500 employees). Due to the relatively limited number of answers from small organizations, it was decided to analyse them together with medium-size organizations.

When analysing the answers originating from *large organizations*, tools and environment-related issues turned out to be even more emphasized, by 94% of all answers, as the most damaging factors likely to undermine the success of distributed software development projects. Communication-related problems were still in second position with 76% of the responses. Issues related to cultural differences were considered equally important as project management issues, making up 65% of the responses. The reasons for these problems were the same as explained earlier in this paper.

The views of *small and medium-size organizations* were slightly different, resulting in a change in the ranking of the most damaging factors. The area of design knowledge was assessed as causing the most trouble by 80% of the respondents. Interestingly enough, in smaller organizations, problems related to cultural differences were mentioned by only 30% of the respondents. The reason for this may be explained by the characteristics of the related distributed projects: the extent of distribution was not as great as that of large organizations. When operating in Europe, cultural differences were not recognized as a major factor, as one respondent referred to this issue as follows: 'Cultural differences are directly proportional to geographical distance of sites. Inside Europe these issues are much easier than between different continents.'

Issues contributing to satisfaction

The survey included a question focusing on the overall satisfaction with the distributed software development that had taken place at the respondents' own company. The following scale was used to make assessment: completely satisfied, somewhat satisfied, somewhat dissatisfied, dissatisfied

and not applicable. The results yielded by this question were very interesting: none of the respondents were completely satisfied, while only 7% were dissatisfied. The majority of respondents (63%) were somewhat satisfied, most of them representing large organizations with over 500 employees. The overall satisfaction rate of respondents from small and medium-size organizations was slightly lower and evenly scattered from somewhat satisfied to dissatisfied. However, the results show that despite the risks and the problems discussed above, the respondents still perceive distribution to be appropriate and, as such, an important and fruitful development strategy.

The overall satisfaction and the issues behind satisfaction or dissatisfaction deserve a more detailed investigation. To clarify possible reasons for this result, the group who expressed themselves to be somewhat satisfied was selected for further analysis. First, the selected answers were compared with the related project duration (by calendar years) and size by person months. However, no clear relationship could be distinguished, although a clear relationship between satisfaction and size of project by the number of persons involved was detected. Figure 9 depicts this relationship. The most satisfied (59%) respondents were involved with projects that employed fewer than 20 persons, while the satisfaction diminishes to 12% as the number of developers rises to over 50. Figure 9 also shows a similar outcome for the categories somewhat dissatisfied and dissatisfied (both classes grouped into the 'Dissatisfied' class in Figure 9: 57% of dissatisfied responses originate from projects involving over 50 persons.

The analysis suggests that the number of persons involved in the project is a significant determinant for the success of distributed software development. Another implication is that since communication-related issues are an important factor determining overall satisfaction then, if the number of persons involved in the project is under 20, this is likely to provide better teamwork-based thinking and practices during the course of the distributed software project. This outcome is worth considering when planning distributed project development, e.g. when making decisions about splitting and conducting project tasks into independent and rational parts at each distributed site.

Limitations of the results

The moderate sample size of 27 allows only limited analysis and study of interdependencies between variables. The possibilities for a more fine-grained analysis are inevitably to some degree limited. The best response rate was achieved when using regular mail (the questionnaire was also sent out using e-mail). The questionnaire was sent to companies that were either known or assumed to have distributed their software development. The mailed questionnaire was received by 101 recipients in 44 companies and of these recipients 24 completed the survey. In total 31 replies were retrieved, of which four were withdrawn from the sample size due to the fact that these respondents were not

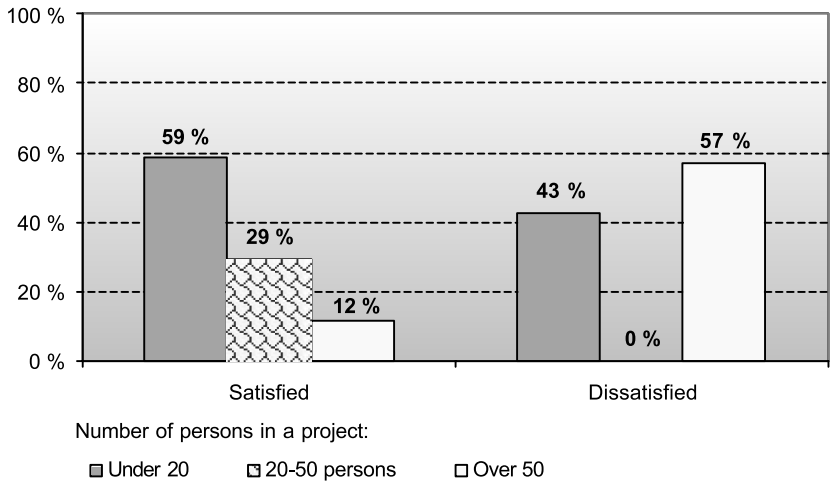

Figure 9 Overall satisfaction versus size of a project

involved with distributed software development and had no experience of it. Despite the fact that the size of the sample is rather small the sampling itself is relevant in the context of this study.

While observing the software development distribution (refer to Table 1 and Figure 3), it is notable that replies were almost equally divided between those that addressed concerns relating to the development of software in a distributed European setting (51.9% of responses) and those that involved distribution between different continents (Europe and the USA and Europe and Asia), the latter contributing 48.1% of responses. However, it has to be acknowledged that replies represent Finnish experience of software distribution to a large extent. This being the case it is reasonable to ponder to what extent the results are generalizable. Recently, Prikladnicki *et al.* (2003) analysed 22 interviews from one Brazilian and one US-based company which had engaged in distributed software development both locally and globally. In their study the conclusions of the relevant problematic issues are consistent with the problem statements of this survey. They name language barriers, communication, cultural differences, context sharing, trust acquisition between teams, requirements engineering, software development process, software configuration, and knowledge management in general, as problem areas in which projects are suffering.

CONCLUSIONS

The challenges of distributed software development must be recognized when aiming at diminishing the risk of development failure and maximizing the possibilities for success. The results of this survey put forward and clarify the target areas for improvement regarding distributed software development processes and suggest appropriate work practices that have been tested in industrial environment.

Software development is very knowledge-intensive field of engineering. In each development phase efficient knowledge creation, knowledge transfer, knowledge storing and/or and knowledge-sharing activities are vital. The results of this survey point out that previously complex software development processes are further complicated by the distribution of the development process over multiple sites. Diverse software development and management knowledge needs and knowledge-capturing and transfer-related problems were identified as being important. In the survey, misunderstandings, ignorance and uncertainty are examples of common words respondents use to describe origins for problems related to requirements engineering, development tools and environment, design knowledge or project management.

Survey results show that the most problematic area is related to software development tools and environment, and more specifically the incompatibility of tools and versions used by the different development sites. This problem is most emphasized by large organizations employing more than 500 persons. Communication problems appear to be extremely common within all organizations; in all, this problem area was ranked as the second toughest. However, a closer analysis of the responses shows that the role of communication is even greater than it appears at first sight. When studying the reasons behind other problems, the lack or poor quality of communication was often mentioned as a root cause. When requesting the three areas where respondents would need support in distributed software development one respondent condensed his answer to the following: 'communication, communication, communication'. When the efficient knowledge-capturing and sharing mechanism (in the form of a face-to-face meeting and personal contacts) is hampered by distance and time differences supportive and substitute solutions are self-evidently vital. Results of the planning phase of a distributed development project are decisive what comes to transferring knowledge between sites. Because knowledge transfer is problematic attempts should be made to minimize it. When the project is divided into independently developed and managed units whose interfaces are clearly defined the needs, for example, for transferring design knowledge during development should decrease.

When knowledge transfer has to take place, appointing a contact person at each site is a means of controlling communication both between sites and within site(s). Numerous communication points at each site raise the risk that knowledge transfer becomes chaotic and uneven.

Requirements engineering and management appeared highly problematic for distributed development projects, causing many errors. Requirements engineering is a large and multidisciplinary process and traditionally is performed at the beginning of the system development life cycle (Royce, 1970). However, in the development of large complex systems it has been realized that it is impossible to define an accurate set of requirements that are likely to remain stable throughout the months or years of development (Dorfman, 1990). Requirements engineering thus becomes an incremental and iterative process, performed in

parallel with other system development activities, such as designing and implementing, which is why misinterpreted, changing and missing requirements are likely to be the main sources for software errors according to the industry feedback.

Based on the analysis described in this article, there is no doubt about the message of the study: successful distributed software development requires both structured—and disciplined—software engineering and knowledge management solutions embodying, most particularly, communication management and the utilization of effective substitutes for face-to-face communication. A careful execution of project start-up activities—including planning (splitting tasks, schedule, delivers), exact determination of common rules, responsibilities, tools used and kick-off sessions—can greatly contribute to a successful implementation. By understanding the nature and demands of distributed software development in depth, software organizations are able to reduce the risk of failure and to make their operations successful.

ACKNOWLEDGEMENTS

This article is based on a survey conducted as a part of the Knots-Q-program (Knowledge-Centered Tools and Methods for Software Production Quality; see Knots-Q Project, 2004), which is funded by the Finnish Academy and VTT, the Technical Research Centre of Finland. The authors are grateful for the voluntary help provided by the representatives of the various organizations involved in this study.

REFERENCES

Aversano L, De Lucia A, Gaeta M, Ritrovato P. 2003. Genesis, 2003: a flexible and distributed environment for cooperative software engineering. In *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 497–502.

Battin RD, Crocker R, Kreidler J, Subramanian K. 2001. Leveraging resources in global software development. *IEEE Software* March/April: 70–77.

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.

De Souza CRB, Basaveswara SD, Redmiles DF. 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, FL.

Dorfman M. 1990. System and software requirements engineering. In *IEEE System and Software Requirements Engineering*, Thayer RH, Dorfman M (eds). Tutorial. IEEE Software Computer Society Press: Los Alamos, CA.

Ebert C, De Neve P. 2001. Surviving global software development. *IEEE Software* **18**(2): 62–69.

Gaeta M, Pierluigi R. 2002. Generalised environment for process management in cooperative software engineering. In *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC)*, 2002. Oxford, UK; 1049–1053.

Gianpalo C, Ghezzi C. 1999. 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*, Stanford, CA; 32–39.

Gorton I, Motwani S. 1996. Issues in co-operative software engineering using globally distributed teams. *Information and Software Technology* **38**: 647–655.

Herbsleb J, Moitra D. 2001. Global software development. *IEEE Software* **18**(2): 16–20.

Herbsleb JD, Mockus A, Finholt TA, Grinter RE. 2001. An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, Toronto; 81–90.

Karlsson E-A, Andersson L-G, Leion P. 2000. Daily build and feature development in large distributed projects. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM Press: Limerick, Ireland; 649–658.

Knots-Q Project. 2004. VTT Electronics. http://www.vtt.fi/ele/research/soh/projects/knots-q/index.htm [23 February 2005].

Kötting B, Maurer FA. 1999. Concept for supporting the formation of virtual corporations through negotiation. In *IEEE Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Stanford, CA; 40–47.

Lam HE, Maheshwari P. 2001. Task and team management in the distributed software project management tool. In *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC)*, Chicago, IL, 401–408.

Lloyd WJ, Rosson MB, Arthur JD. 2002. Effectiveness of elicitation techniques in distributed requirements engineering. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*. 311–318.

Mockus A, Herbsleb J. 2001. Challenges of global software development. In *Proceedings of the 7th IEEE International Software Metrics Symposium (METRICS 2001)*, 4–6 April, London. IEEE Computer Society; 182–184.

Niederman F, Beise CM, Beranek PM. 1993. Facilitation issues in distributed group support systems. In *Proceedings of Conference on Computer Personnel Research*. ACM Press: New York; 299–312.

Press L. 1993. Software export from developing nations. *IEEE Computer* December: 62–67.

Prikladnicki R, Audy JLN, Evaristo R. 2003. Global software development in practice lessons learned. *Journal of Software Process Improvement and Practice* **8**(4): 267–279.

Royce WW. 1970. Managing the development of large software systems. *Proceedings of IEEE Wescon*. Reprinted in *Proceedings of the 9th International Conference on Software Engineering* (1987), Los Alamitos, CA. IEEE Computer Society Press; 328–338.

Surjaputra R, Maheswari P. 1999. A distributed software project management tool. In *IEEE Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, USA.

Van der Hoeak A, Heimbigner D, Wolf AL. 1996. Peer- to-peer repository for distributed configuration management. In *Proceedings of the 18th International Conference on Software Engineering (ICSE '96)*, Berlin; 308–317.

Zoran M, Berry A, Bond A, Raymond K. 1995. Supporting business contracts in open distributed systems. In *Proceedings of SDNE '95*. IEEE Computer Society Press: Whistler, Canada.

PAPER II

How to build and sustain a measurement data management environment in a SME

In: Proceedings of Software Measurement European Forum (SMEF 2006). Rome, Italy.
Pp. 225–236.

Published in the Proceedings of Smef 2006

How to build and sustain a measurement data management environment in a SME

Maarit Tihinen, Janne Järvinen

Abstract

Measurement is seen as a valuable way to get information and knowledge of a company's software development processes and possible improvement needs. However, measurement is still experienced as a problematic and troublesome activity in the software production. This paper shows that there are numerous standards, practices, methodologies and approaches affecting the measurement process in an organisation. The number of possibilities may appear confusing, especially because SMEs (Small and Medium-sized Enterprises) cannot afford wrong decisions. In addition, there are many other challenges for building and sustaining a measurement environment in the SME organisations; for example, limited resources have been tied to "the real work", and the added value of the measurements is difficult to prove.

The purpose of this paper is to describe elements of measurement data management (MDM) which have to be considered when building and sustaining a measurement framework in a SME. This paper introduces step by step how MDM framework was built and also what kind of experiences – weaknesses and strengths – were observed when MDM was built and sustained in a SME. Shortly, MDM covers both organisation and project level processes, tools and solutions for defining metrics, gathering, controlling and analysing measurement data, aiming at a better understanding of processes and existing practices or improving them.

1. Introduction to the MDM framework

The measurement is still experienced as a problematic and troublesome activity in the software production [1]. This paper shows that there are numerous standards, practices, methodologies and approaches that affect the measurement process in an organisation. Amount of possibilities may feel confusing, especially because SMEs cannot afford wrong decisions. Also, there are many other challenges for building and sustaining measurement environment in the SME organisations, for example, limited resources have been tied to "the real work", and gained add-value of measurements is difficult to prove. This paper proves that it's possible to build and sustain a successful measurement data management (MDM) environment for a SME organisation.

This paper introduces, step by step, how the MDM framework was built and also what kind of experiences – weaknesses and strengths – have been observed when MDM has been built and sustained in a SME. The MDM framework is introduced with a case study where the MDM solution has been successfully implemented at Solid Information Technology Corp. [2], henceforth Solid. Solid is a SME that provides an advanced data management platform for embedded products, such as network infrastructure, fleet management, consumer electronics and digital home. VTT, Technical Research Centre of Finland [3], has experience in applying software measurements in several process improvement projects with national and international companies in embedded software engineering area. This paper is structured as follows. In this section MDM framework processed by VTT and its theoretical background are considered. In the second section, a case study where MDM has been applied for SME organisation is introduced by following an identified list of MDM issues. At the end of the paper, central observations and experiences are concluded.

MDM covers both organisation and project level processes, tools and solutions for defining metrics, gathering, controlling and analysing measurement data, aiming at a better understanding of processes and existing practices or improving them. MDM is a framework enabling several approaches, techniques or standardised processed for measuring by indicating the most vital aspects from the measurements point of view. The importance of the measurements culminates in activities related to software product quality (project level) and software process improvement (organisation level), shown in Figure 1.

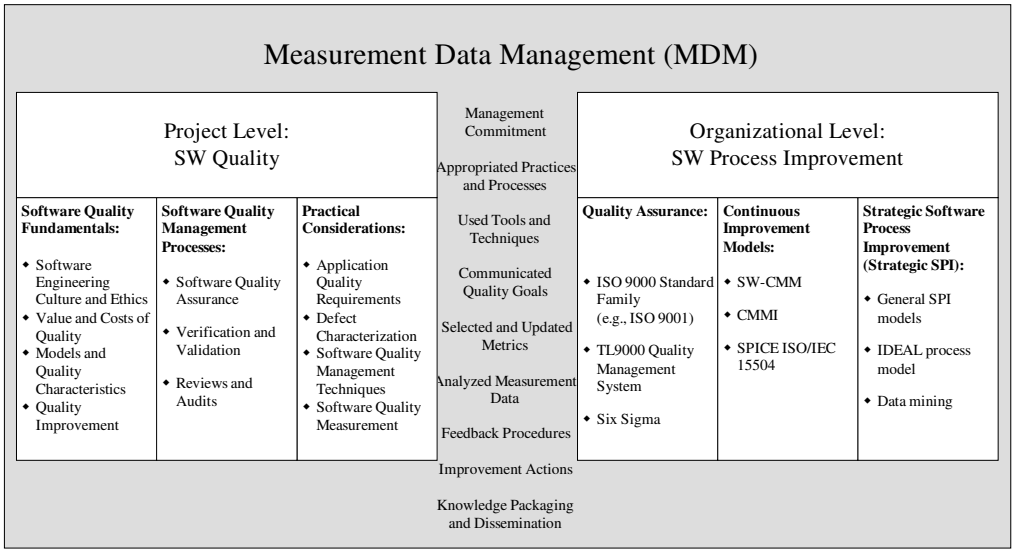

Figure 1: Elements of the MDM

Project level; Software Product Quality: At the project level, many practices and processes are defined in separate standards. The SWEBOK [4] introduces that Software Quality Key Area can be divided to the following topics: 1) software quality fundamentals, 2) software quality management process and 3) practical considerations. In addition, the SWEBOK emphasizes that software quality is a ubiquitous concern in software engineering. The ISO/IEC 9126 standard [5] defines six quality characteristics and several sub-characteristics with associated metrics and describes a software product evaluation process model, the ISO/IEC 12207 standard [6] defines software quality management processes (quality assurance process, verification process, validation process, review process, audit process), and the ISO/IEC 15288 [7]defines life cycle processes of system engineering, for example. In addition, The IEEE 1061 standard [8] is a systematic approach to establishing quality requirements and identifying, implementing, analysing and validating the process and product of software quality metrics for a software system, while the ISO/IEC 15939 standard [9] comprehensively determines the software measurement process in software engineering. Furthermore, the project level practices and processes reflect the decisions that have been made at the organisation level, e.g., the chosen and certified quality management system, conformed maturity requirements of selected assessment models or adopted measurement approach. For example, the GQM paradigm [10] [11] represents a systematic approach for tailoring and integrating the objectives of an organisation into measurement goals and their stepwise refinement into measurable values.

Organisational level; Software Process Quality: The software process improvement activities can be considered from three viewpoints: 1) quality assurance, 2) assessment models and 3) strategic software process improvement (Strategic SPI) models, adopted from [12][13][14][15].

Quality assurance covers the organisation's certified quality management systems (QMS), such as ISO9001 [16] or TL9000 [17], but also the uncertified organisational quality assurance systems or practices and process descriptions. The most important point of those practices and systems is to harmonize both the organisational and project level activities and processes through a company. In addition, there are several improvement methods available that recommend the use of measurements to evaluate the usefulness of the improvements. Examples of such methods are QIP [10], SigSigma [18], Pr2imer [19] and PROFES [20].

Software processes can be evaluated by using SW-CMM [21], CMMI [22] or ISO/EIC 15504 [23], for example. The models guide the developers in gaining control of their development and maintenance processes and in evolving towards a culture of software engineering and management excellence. The maturity structure helps software professionals and managers to identify areas where improvement actions will be the most fruitful. While quality management systems identify improvement approaches, continuous improvement models (like CMMI or SPICE) identify practices that should exist in the organisation [24]. The assessments are typically used for finding process areas or activities that should be systematically improved.

The strategic software process improvement (strategic SPI) models can be used for identifying how the improvement issues should be implemented in a target organisation. From measurement view of point, the strategic SPI also covers means like data mining or ODC analysing [25] that can be used for obtaining deeper knowledge of current practices and processes.

Measurement Data Management (MDM) framework: The main goal of MDM is to integrate both project and organisation level measurement activities for producing an environment of comprehensive measurement utilisation. The following MDM framework issues should be taken into consideration one by one while building an organisation specific solution:

- Management Commitment.
- Appropriate Practices and Processes.
- Used Tools and Techniques.
- Communicated Quality Goals.
- Selected and Updated Metrics.
- Analysed Measurement Data.
- Feedback Procedures.
- Improvement Actions.
- Knowledge Packaging and Dissemination.

To be successful, a measurement utilisation requires a firm support – commitment - from the organisation's management. The organisation strategy and vision, and the chosen quality assurance or assessment models influence the appropriate practices and processes, also while describing or defining them. Also, commitment to measurement throughout the organisation is of utmost importance, and can be ensured by management activities and decisions.

While tools and techniques should be taken into account while describing processes and activities, they should also be studied from the viewpoint of measurements: e.g., tools can enable automatic data collection and analysing. In addition, a comprehensive solution may require needs for purchasing new tools or techniques. Needs for manual data collection and, thus, needs for new templates or modifications to the available templates, etc. should also be taken into account. Furthermore, maybe the most challenging aspects in measurement environment are how to put the feedback procedures, improvement actions and knowledge sharing into practice. It is important to realise that MDM is a general framework and a detailed solution should be built step-by-step and adapted to organisation's own needs and quality goals communicated both on the project and the organisational level.

2. Case study – applying MDM for a SME environment

In this section a case study of how MDM was applied in a SME organisation is described in detail. Each of the MDM framework issues has been studied when building the measurement environment in Solid.

2.1. Management commitment

Generally, a successful measurement program requires strong commitment from the organisation's management. In Solid, there was a SPI (software process improvement) board that set quality goals for measurements. The SPI board consisted of the CTO (Chief Technology Officer), Director of R&D, Lead Architect and the SPI manager. The measurement strategy was derived from Solid's business strategy and defined by the SPI board as follows: "*Measurement strategic intent is to collect and use measures that help us to determine and maintain the quality of our products and to continuously monitor that our SW development processes are effective and efficient*".

2.2. Appropriate practices and processes

In Solid, all process descriptions, general practices and guides are available for the staff via Intranet. Thus, Solid's measurement environment was also planned to be used via Intranet to being available when needed. In addition, because of the limited resources of the SME organisation, it was a natural aim to build measurement environment as automatic as possible.

2.3. Used tools and techniques

The used tools are in a central role when considering what kind of data and information it is possible to get out automatically. Although tools should not be in a dominant position but they can limit the possibilities and/or act as a preventive factor. That is why it is necessary to examine the used tools and techniques for understanding their possibilities, for example:

- What kind of data and data items they produce.
- Where data items are saved (data storages, databases).
- How data can be accessed and manipulated.

In Solid, the following software development related tools are used for measurement purposes:

- Lotus Notes [26], used for handling of external problem reports.
- Buzilla [27], used for handling of internal bugs and errors.
- Perforce [28], used for version management.
- Planmill [29], used for planning and controlling of effort, cost and schedule.

The measurement data is automatically collected from version management system (Lotus Notes and Perforce), project management system (Planmill) and defect management system (Bugzilla). Solid's measurement environment covers tools that enable automatic data collection shown in Figure 2.

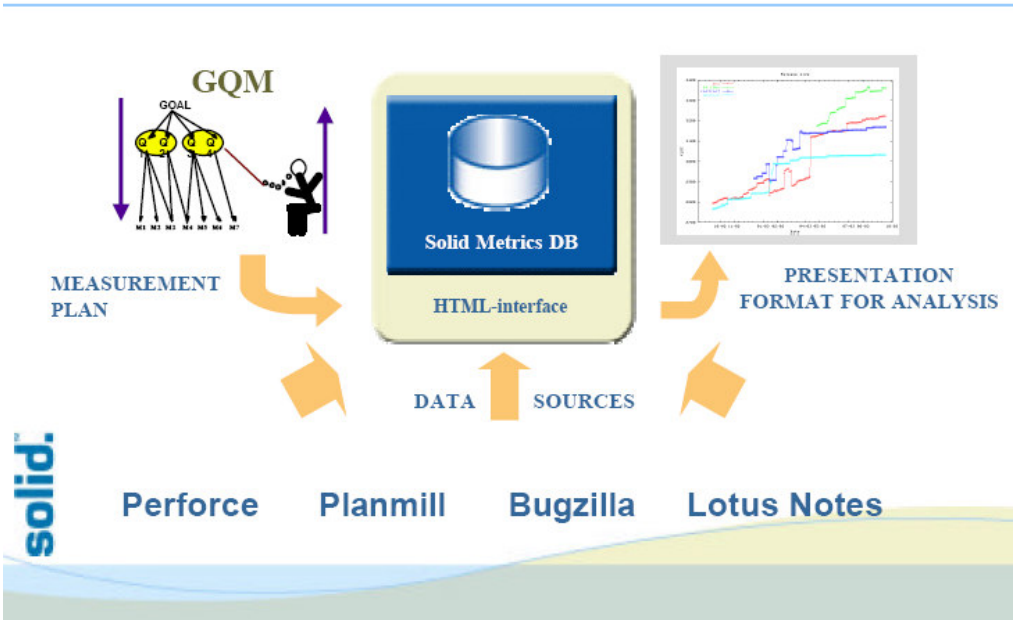

Figure 2: Solid Measurement Environment

2.4. Communicated quality goals

According to the Solid strategy, the following quality goals were identified to be the most important to Solid's success from R&D point of view:

- Product Reliability.
- SW Development Predictability.
- Product Portability.

Solid's SPI board set the quality goals, thus the link between the business strategy and the quality goals were ensured. Also other quality goals were identified and those are to be implemented during the next iterations of the measurements improvements. A specific interest was to build traceable links between business goals and everyday activities of R&D engineers. This made quality goals understandable and meaningful.

2.5. Selected and updated metrics

The GQM (Goal-Oriented-Measurement) paradigm was used for defining metrics for each identified quality goal. The following table (Table 1) describes what kinds of metrics were defined for monitoring the "Product Reliability" quality goal, for example.

Table 1: An example of metrics definition by using the GQM method

GOAL: Product Reliability

Measurement Goal:

Analyze the **product and process** for the purpose of **understanding and control** with respect to **reliability** from the viewpoint of **project team and management** in the context of Solid SW development.

Questions and Metrics

Q1: From the customers' viewpoint, is the reliability of the Solid products increasing?

M1.1 Product Release Defect Trend
M1.2 Release Size
M1.3 Release Defect Density
etc.

Q2: How are different test phases affecting to reliability?

M2.1 Cumulative Release Defect Count
M2.3 Project Testing Effort Distribution Over Time
M2.4 Project Defect Class Profile
M2.5 Release Test Coverage
M2.8 Release Testing Success Rate
etc.

After metrics definition, each metric and data elements were described using templates (shown in Table 2) processed by Tihinen [30].

Table 2: Template for metrics description [30]

NAME OF THE METRIC	
GENERAL	
Purpose:	
Characterization (classification):	
GOAL AND USAGE	
Quality goal	
Use in project	
Use in organisation	
COLLECTION	
Data items	
Tools	
Who	
When	
Procedure	
ANALYSIS	
Presentation options	
Indicator example	Figure.
Calculation	
Interpretation	
Associated measures	
Target value	
When (Who)	
Tools used to generate indicator	

The used template forces one to define the procedures of data collection and results presentation and interpretation. The template guides one in determining an indicator example and how the indicators should be interpreted.

All this – metrics descriptions, indicator examples, result graphs, interpretation guides, associated metrics, etc. – were implemented in the Solid's Intranet. A basic aim of Solid's measurement environment is that data collection and indicator (/graph) producing are done as automatically as possible. A special concern was to keep measurement environment simple and easy to maintain.

2.6. Analysed measurement data

All metrics indicators are produced automatically to Intranet by using scripts. The collected measurement data is stored into the Solid metrics database from where automated scripts produce the results in a visible form, e.g., graphs and bars. Typically, a script is run once a day and the data is sent to the metrics database. Then, another script refreshes the data in the Intranet. Ease of use of scripts was highly prioritised; for example, there is a link for checking how a graph has been built, and also modification of scripts is possible (scripts are under version control). The following figure shows an indicator example of "Product release defect trend". Figure 3 shows the numbers (#) of the problem report items that have been assigned to R&D after 120 days of the release date.

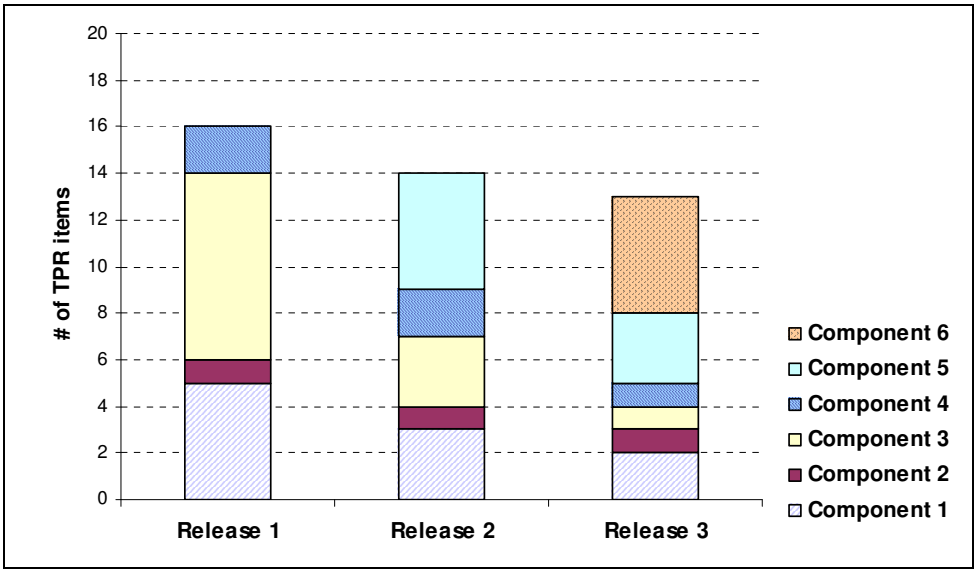

Figure 3: An indicator example of metric: Product release defect trend

Also, guides that help interpreting the graphs and observed things that can distort the figures are also included in the Intranet for each metrics description. In addition, associated metrics that can help interpretation are mentioned and introduced (see Table 3). Naturally it is the experts who finally decide what interpretation makes sense in given circumstances.

Table 3: An indicator example of the interpretation and mentioned associated metrics

Metric: Product Release Defect Trend

Interpretation:

For the reliability to increase, the number of defect found from the latest version should be lower than what it has been in the previously released versions. If there seems to be a negative trend, further analysis is required. It might be useful to take a look at the component defect density to isolate the problem area. To estimate how the defect curve will develop after the product release, the cumulative number of defects can be split to, for example, one week time periods and compare the curve to the ones of previous releases.
Things that can distort the defect figures:

- Some parts of the product are not used as heavily as others → defects in extensively used parts show up earlier and the probability of finding is bigger that in the less used parts
- Typically, size of the product is increasing in each new release → assuming that the defect injection rate remains the same, the total number of defects increases as the product gets bigger
- On the other hand, if large parts of previously released software is reused, most defects have already been found -> number of defects go down

The effect of these issues must be considered in the analysis of the results.

Associated Metrics:
Following metrics help interpret this metric

- Release Size
- Release Defect Density

2.7. Feedback procedures

In the project level feedback procedure is straightforward as the metrics and graphs are available in the Intranet. The automated data presentation procedure (scripts) makes it possible to get and give up-to-date information during project lifecycle. Thus, both project managers and members have a choice to influence practices, for example, by studying reasons or making corrective and improvement actions.

In the organisational level feedback procedure was not so easy to get working systematically. Several metrics reports were defined to be produced in the predefined schedule. In practice, however, operational priorities often delayed feedback processing. .

2.8. Improvement actions

The Intranet based measurement environment enables a fast response for deviations or problems shown by the analysed results. Project managers and engineers can control and follow the situation on real-time. This is well suitable to the SME organisation as improvement actions can also be done immediately.

The built measurement environment enables the analysing of products and processes in Solid. For example, in the Intranet, daily updated bar graphs and trend presentations are available. Besides the continuous process analysing, Forschungszentrum Informatik (FZI) [31] has assessed Solid's database product. The product assessment contained metrics as follows:

- Architecture violations.
- Measurement of coupling, encapsulation and complexity.
- Complexity and coupling of data objects.
- Complexity and call dependencies of functions.

Although the assessment proved that Solid's product was in a good shape, the summary report also pointed at areas that had to be carefully checked by experienced developers. As a result, corrective actions were planned and implemented. A subset of the FZI product assessment tools was installed in Solid R&D lab, and new versions of Solid have been investigated using these tools.

Naturally, there are always activities, e.g., new quality goals, new or updated metrics, new assessment or monitoring actions, which can be added to the Solid's MDM solution. For example, from the viewpoint of strategic SPI (illustrated in Figure 1) there are needs for examining new improvement actions by measurement data mining and further analysing of the results.

2.9. Knowledge packaging and dissemination

In SME organisation knowledge packaging and dissemination procedures should be as light as possible. The automated data collection procedure ensures that the metrics database is saved and the historical data is available when needed. In addition, the informing of the metrics results can be done face-to-face in a SME. Generally, there is always a danger that knowledge remains inside the heads of a few active persons and that knowledge is not saved anywhere. Solid's measurement environment minimises this problem as it makes it possible to add information (e.g. interpretations and target values) that can be utilised beyond specialists. For example, Figure 4 shows the severity 1 (i.e. serious) post-release defects for a Solid DB product between 2000 and 2005. Post-release defects were a key measure for product reliability that was a major quality goal. Figure 4 shows a clear improvement since 2002 when systematic SPI and measurement programme was introduced.

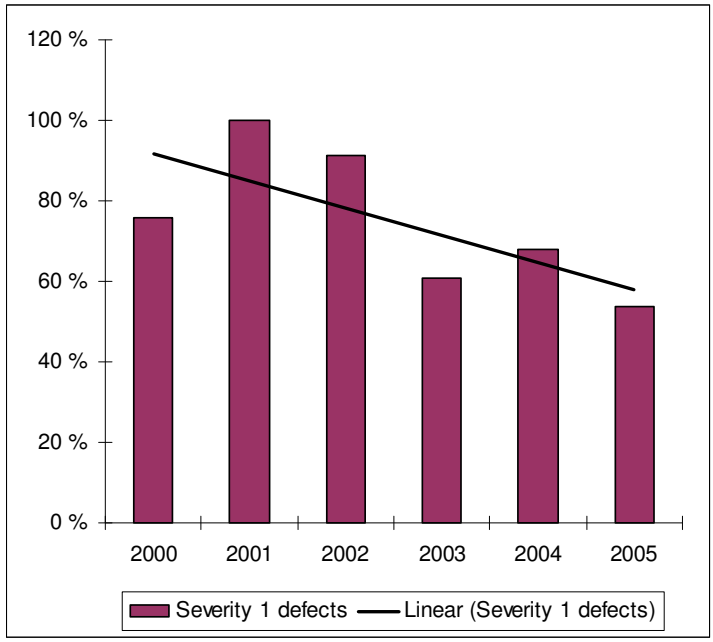

Figure 4: Solid Severity 1 Post-Release Defects in 2000-2005

3. Conclusion

This paper shows that it is possible to build and sustain a MDM also in a SME organisation. The paper points out the basic issues that should be taken into consideration in order to achieve a successful solution. The base is the management commitment; it gives the possibility for a successful building of a MDM framework. Measurements are considered more significant if the goals are derived from the business strategy and goals. Solid's SPI board was in a central role in setting the quality goals. However, the link between business goals and practices can be easier to find in a SME organisation than in a bigger one. In the case of Solid the GQM method was used for deriving practical metrics from the defined quality goals.

The building of the infrastructure (HTML –pages, determining data items, scripts for collecting and analysing data, etc.) needs time and effort. The building can be done step by step, for example, by starting with one quality goal at a time. However, automated data collection seems to be the only choice for a SME organisation to make sure that the measurement data is really stored and available. That is also good to recognise if one is selecting new tools for use. For example, open source tools (e.g. Bugzilla) can often be more easily modified further and integrated with other tools than commercial tools. Measurement data validation, however, always needs manual care and effort.

Automated data collection, scripts for visualising results and Intranet based MDM environment enable data analysing and also improvement actions. In a SME organisation corrective actions can be done flexibly during the project work. Instead, extra effort and resourcing needed attention, for example, detailed data analysing and systematic knowledge packaging can be ignored because their ROI (return on investment) is not so easy to point out.

The paper proved that the built MDM environment is a base for systematic and continued measurements in a SME. The MDM framework enables various methods for measuring; for example, FZI assessments gave added value for assuring a product quality in the Solid case. In addition, the built MDM creates a base for strategic SPI activities and further analysing; e.g., ODC analysing for achieving deeper knowledge of current practices and processes is possible as the historical measurement data is available. However, these kinds of activities require extra effort and resources, and so these activities are usually passed in SME.

This paper described elements of MDM, which have to be considered while building and sustaining a measurement framework in a SME. Even if the limited resources have been tied to "the real work" and the added value of the measurements is difficult to prove, it is possible to build and sustain a successful MDM environment for a SME organisation.

4. References

[1] Rifkin, S. "What makes measuring software so hard?" IEEE Software, May/June 2001, pp. 41-45.

[2] Solid Information Technology corporation. Home page URL: http://www.solidtech.com (accessed at 27.02.2006)

[3] VTT, Technical Research Centre of Finland. Home page URL: http://www.vtt.fi/indexe.htm (accessed at 27.02.2006)

[4] Abran, A., Moore, J. (Co-executive Editors), Bourque, P., Dupuis (Co-Editors) R., Tripp, L., "Guide to the Software Engineering Body of Knowledge – 2004 Version - SWEBOK", IEEE Computer Society Press, April 2005, pp 200, ISBN 0-7695-2330-7.

[5] ISO /IEC 9126:2001. "Software Engineering – Software Product Quality evaluation" (Quality characteristics and guidelines for their use), International Organisation for Standardisation.

[6] ISO/IEC 12207:1995 /AMD2:2004. "Information technology – Software life cycle processes", International Organisation for Standardisation.

[7] ISO /IEC 15288:2002. “System engineering – System life cycle processes”, International Organisation for Standardisation.
[8] IEEE Std 1061. “IEEE Standard for a software quality metrics methodology”, (ANSI) The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey. 1992.
[9] ISO/IEC 15939. “Software engineering -Software measurement process”, International Organisation for Standardisation. 2002.
[10] Basili, V., Caldiera, G. and Rombach, H. “Goal Question Metric Paradigm”. In John J. Marciniak, editor, Encyclopaedia of Software Engineering, Vol 1, John Wiley & Sons, 1994, pp. 528–532.
[11] Solingen, R. and Berghout, E. ”The Goal/Question/Metric (GQM) method, a practical method for quality improvement of software development”, McGraw-Hill, 1999.
[12] Järvinen, J. Doctoral Thesis: “Measurement based continuous assessment of software engineering processes”, Espoo, Finland, VTT Publications 426. 2000. 97p. + app. 90p.
[13] Karlström, D., Runeson, P. and Wohlin, C. “Aggregating viewpoints for strategic software process improvement – a method and case study”. IEEE Proceedings-Software, Vol. 149, No.5, October 2002, pp. 143-152.
[14] Komi-Sirviö, S. Doctoral Thesis: “Development and evaluation of software process improvement methods”. Espoo, Finland, VTT Publications 535. 2004. 175p. + app. 78p.
[15] Varkoi, T. “Management of Continuous Software Process Improvement”. Engineering Management Conference (IEMC '02), Vol 1, 2002, pp. 334 – 337.
[16] ISO 9001:2000. Quality Management Systems – Requirements. December 2000.
[17] TL 9000 Telecommunications Industry Standard. 1999.
[18] SigSigma home page URL: http://www.sigsigma.com/ (accessed at 27.02.2006)
[19] Mäkäräinen, M. and Komi-Sirviö, S. “Practical process improvement for embedded real-time software”, proceedings of the 5th European Conference on Software Quality, Dublin, IR, 16 - 19 September, 1996, pp. 408 – 416.
[20] Birk, A., Järvinen, J., Komi-Sirviö, S., Kuvaja, P., Oivo, M., and Pfahl, D. ”PROFES - A product driven process improvement methodology”, European Conference on Software Process Improvement (SPI '98), John Herriot. Monaco, 1 - 4 December 1998. 9 p.
[21] SW-CMM. Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. “Capability Maturity Model for Software”, Version 1.1. Software Engineering Institute, Technical report CMU/SEI-93-TR-24. 1993.
[22] CMMI. Capability Maturity Model Integration (CMMISM) for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing, version 1.1 (CMMI-SE/SW/IPPD/SS, V1.1), Staged Representation, CMU/SEI-2002-TR-012, March 2002.
[23] ISO /IEC 15504:2003. Information technology -Software Process Assessment. Parts 1-5.
[24] Card, D.N. “Research directions in software process improvement”, In the Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004 (COMPSAC’04). Vol 1. pp. 238.
[25] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K. and Wong, M-Y. “Orthogonal Defect Classification – A concept for In-Process Measurements”, IEEE Transactions on Software Engineering, Vol.18, No.11, November, 1992.
[26] Lotus Notes home page, URL: http://www-306.ibm.com/software/lotus/ (accessed at 27.02.2006)
[27] Bugzilla home page, URL: http://www.bugzilla.org/ (accessed at 27.02.2006)
[28] Perforce home page, URL: http://www.perforce.com/ (accessed at 27.02.2006)
[29] Planmill home page, URL: http://www.planmill.com/ (accessed at 27.02.2006)
[30] Tihinen, M. Secondary subject thesis: “Analysis of the quality measurement processes in software production”. University of Oulu. 42 p + app. 7p. 2001.
[31] FZI. Forschungszentrum Informatik Home page, URL: http://www.fzi.de/eng/index.php (accessed at 27.02.2006)

PAPER III

ABB experiences of boosting controlling and monitoring activities in collaborative production

In: Proceedings of the Sixth IEEE International Conference on Global Software Engineering, 15–18 Aug. 2011, Helsinki. 5 p.

ABB Experiences of Boosting Controlling and Monitoring Activities in Collaborative Production

Maarit Tihinen, Päivi Parviainen, and Tanja Suomalainen
Software Technologies
Technical Research Centre of Finland, VTT
Oulu, Finland
firtsname.lastname@vtt.fi

Katja Karhu and Malin Mannevaara
Electrification, Instrumentation and Composite Plants
ABB
Vaasa, Finland
firtsname.lastname@fi.abb.com

***Abstract*—In globally distributed projects, successful product development requires companies to pay special attention to project management and controlling activities during the projects' life cycle. If controlling activities are not done well, the productivity of product development decreases. This is due to, e.g., unnecessary work caused by unclear responsibilities, missing information, and not managed dependencies. This paper highlights ABB experiences in boosting globally distributed project management activities by integrating those actions to the portal of the company. The case showed that the implemented reporting practices together with other improvement actions such as enhanced process descriptions, practices and engineering tools, reduced the unnecessary or free work done at ABB by even 45%. This successful result was achieved by careful current state analysis where the requirements and goals for improvement actions were defined. The improvement actions were defined by discussions with essential interest groups as well as considering the best practices and research results from the literature.**

Keywords-project management; collaboration management; global product development

I. INTRODUCTION

Project management covers the activities like planning, scheduling, organizing, controlling, and managing tasks and resources to achieve the successful completion of a set of specific project goals. Nowadays, when the products are increasingly developed globally, in collaboration between subcontractors, third party suppliers and in-house developers [1], proper project controlling and monitoring activities are more and more important. Based on an industrial survey [2], one of the most important topics in the project management problem area, in distributed software development, is detailed project planning and strict control during the project. In global software development (GSD), this means, e.g., dividing work by sites into sub-projects, clearly established rules, defined responsibilities, results and timetables along with regular meetings and process controlling.

Distributed, collaborative product development forces to pay attention to management and controlling activities for creating awareness of distributed activities within the project and over the projects in an organization. Modern technology has improved and made possible to convert and deliver project-based information, e.g., by using World Wide Web (WWW) based solution to monitor information regardless of time and location [3]. Today companies can purchase commercial tool solutions or alternatively develop their own tailored systems to track project progress according to their goals and needs. The definition of current state as well as needed improvement actions are the first steps for development actions.

This paper describes experiences of a successful case executed at ABB (http://www.abb.com/) during the ITEA PRISMA (2008-2011) project [4]. ABB is a global leader in power and automation technologies; it operates in more than 100 countries and has offices in 87 of those countries and has about 117 000 employees. The case focused on creating and improving project monitoring, controlling, and reporting practices to attain better coordination between and within the projects as well as to build organization wide awareness of the status of the project portfolio at ABB. Controlling and monitoring activities had been found challenging: most of the projects at ABB are done in a globally distributed way, i.e., to different ABB sites and to suppliers, and all the collaborating sites are not technologically advanced but can be rather basic (e.g., developing countries without internet access).

The paper is structured as follows. Firstly, the case is introduced giving the background analysis of the goals and requirements setting of the case. Also, a brief description of the main literature studies is introduced in section II. Then, the execution of the case with identified requirements and detailed examples of the implementation are described, in section III. After that, the achieved experiences and results are analysed. In the analysis, the strengths and weaknesses of the case and also the business influence have been presented. Finally, the conclusions are drawn in section V.

II. CASE DESCRIPTION

The case was conducted at ABB in the unit of Electrification, Instrumentation and Composite Plants during the ITEA PRISMA (2008-2011) project. First, a current state analysis was conducted to identify challenges faced at ABB in globally distributed projects. As part of the analysis, a survey was sent to all project engineers in May 2009. The purpose of the survey was to find out how much unnecessary or free work was done and what were the major causes for

978-0-7695-4503-5/11 $26.00 © 2011 IEEE
DOI 10.1109/ICGSE.2011.12

that work. In order to define the improvement actions workshops as well as a literature review was conducted.

The literature review was twofold: it included both research articles from the field of the study (i.e. project monitoring and controlling, management reporting, and multi-project monitoring) and solutions from SameRoomSpirit Wiki [5]. The wiki was used in the literature review, since it provides solutions especially to the global software development, and global distribution was found significantly important in ABB's projects.

After, the improvement actions were defined, the case was implemented by taking the actions into use. Thereafter, the first survey was repeated in order to verify the effectiveness of the actions executed. The current state analysis and the literature reviews are defined in more detail next. Thereafter, case scoping is presented.

A. Identified challenges at ABB

ABB had identified problems relating to project and supply management. For one, the customer had different interpretations than ABB and thus change management during the project was difficult. All gaps and overlaps would create extra work which probably would not be paid by the customer as the delivery limits were unclear (this is called "free work"). Also, all customer requirements and specifications were not forwarded to supplier or the scope of supply was unclear. Based on the problems, "unnecessary work" was defined. In ABB case "unnecessary work" means time used for just waiting or doing duplicate work or unclear tasks, for example.

The first step to find the right focus for improvements was to investigate how much unnecessary work was done, and what were the major causes for the unnecessary work. A survey was sent in May 2009 to all project engineers via electronic Intranet Survey tool. It was also agreed that the effectiveness of the executed improvement actions would be monitored and verified by repeating the same survey after the actions had been implemented. The first survey showed that even 19% of the project time was wasted to unnecessary or free work. Thus, workshops were arranged to analyse reasons behind that: the "five why" -method [6] was used for clarifying the root causes of the unnecessary and free work.

The analysis of the reasons pointed out that a major part of the unnecessary or free work was caused by that the current process descriptions were not followed or they were out-of-date or not applicable to all projects. Also, there was no common systematic reporting practice. As a main result it was agreed that better coordination between and within the projects was needed and that following the process instructions should be ensured. The improvement actions would be focused on the coordination and monitoring needs of project members, managers and different stakeholders during distributed projects.

B. Proposed by SameRoomSpirit Wiki

During the literature study guidelines and solutions from SameRoomSpirit Wiki [5] were researched in order to take the distributed development viewpoint into account well. SameRoomSpirit Wiki is a wiki-based tool developed during the PRISMA project and it provides access to relevant solutions about global software development (GSD) and experiences of their use. The solutions are created based on industrial experiences and literature studies. Based on the solutions from SameRoomSpirit Wiki, the following general advices regarding to better reporting practices were found useful:

1. Distributing drafts of proposed schedules and task assignments for each incremental release [7].
2. Weekly task reports [8].
3. Delivery reports (description of the changes /features that are checked in) [8].
4. Quarterly sync-up meetings (face-to-face) [8].
5. Revising all documentation and updating documents to reflect the current state of the development [9].
6. Informing and monitoring should be followed-up in all directions. Parties should comment all points in the follow-up report. The follow-up reports include tasks done, open questions, problems, and future outlook [10].
7. Also weekly meetings inside a subgroup were proposed [11, 12].

Although, the above practices may seem quite general, they need special attention in globally distributed context.

C. Literature proposals for project monitoring and controlling

Several literature sources were studied in order to get input from project monitoring and controlling viewpoint, especially, in global context. The most relevant sources from ABB situation viewpoint were CMMI and [13] which are briefly presented next. CMMI® (Capability Maturity Model® Integration) models are collections of best practices that help organizations to improve their processes. CMMI for Development (CMMI-DEV) [14] provides a comprehensive integrated set of guidelines for developing products and services. The Project Monitoring and Control (PMC) process area is one of the maturity level 2 process areas of CMMI-DEV. The purpose of the PMC process is to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan. In the following table (TABLE I.) the summary of specific goals and practices of the PMC has been introduced.

TABLE I. SPECIFIC GOALS AND PRACTICES BASED ON CMMI [14]

Project Monitoring and Control (PMC) process	
Specific Goal: Monitor the Project Against the Plan	**Specific Goal: Manage Corrective Action to Closure**
- Monitor Project Planning Parameters - Monitor Commitments - Monitor Project Risks - Monitor Data Management - Monitor Stakeholder - Involvement - Conduct Progress Reviews - Conduct Milestone Reviews	- Analyze Issues - Take Corrective Action - Manage Corrective Actions

According to [13], concentrating on the main issues is the key to success with multi-project monitoring and reporting. Multi-project monitoring lies between strategic project monitoring and individual project monitoring. Its aim is to optimize the management of the project portfolio in accordance with divisional and corporate goals. They [13] classified the reporting to the three categories: 1) Decision-based regulatory reporting with decision-support information. 2) Standard reporting for other organizational units and target groups. 3) Key Performance Indicators (KPI) and other indicators. Furthermore, they proposed that to support monitoring processes and enable rapid decision-making, the management reporting must fulfil the requirements presented in the following table (TABLE II.).

TABLE II. THE REPORTING REQUIREMENTS [13]

R#	*Requirements descriptions*
R1	Integrate processes and systems to obtain data in real time.
R2	Reduce administrative effort to obtaining, prepare and supply data.
R3	Enable findings to be retained and reused as decision-support
R4	Streamline decision-making processes
R5	Provide also non-financial performance indicators.

D. Decisions and Case Scoping

As the status of the whole distributed project portfolio was to be followed, the most relevant practices proposed by SameRoomSpirit wiki were 4, 5 and 6 (see section II /B). In addition, the item 2 was recognized as good practice, even if weekly reports were not matching ABB's maturity level and the scope of the projects. In addition, specific practices advised by CMMI-DEV were taken into consideration. Thus, it was decided to consider the issues of ABB as follows:

- Establish monthly synchronization meetings, where the project managers and line managers are synchronizing their activities. The meetings are organized mostly as face to face meetings, however, some project managers need to attend the meeting also remotely, i.e. virtual meeting practices are required.
- Request that all project documentation should be updated regularly.
- For the project follow-up meeting, establish new monthly reporting templates.
- Reporting templates should contain items related to tasks done, open questions, problems and future outlook.
- Establish a new project follow-up template into the project portal, so that it is easy for everybody to follow the project portfolio status also remotely from any location where our projects are carried out.

While identifying in more details the content of new reporting templates the categorised reporting issues by [13] were taken into consideration. It was decided that the items "Standard reporting for other organizational units and target groups" and "Reporting KPIs and other non financial indicators" shall be conformed. Thus, Project Monthly Reporting was decided to call as a standard reporting for other organizational units, and Quality Assurance (QA). Reporting was targeted as reporting KPIs and other non-financial indicators. In addition, the reporting requirements (TABLE I.) were approved to be met while implementing the monthly follow-ups.

III. CASE IMPLEMENTATION

Based on the current state analysis, it was agreed that better coordination between and within the projects was needed as well as assuring that process instructions would be followed. Thus, it was decided to support line management and the rest of the distributed organization with better project monitoring and controlling practices.

A. Implementation

The implementation of the reporting templates started with creating an example content for the reports (Monthly Project Report and QA report), first in Microsoft Excel, and then implementing a prototype into the ABB's collaboration portal. The link to the examples was sent to all project managers for commenting, and also it was reviewed in small review meeting among the people who sent the comments.

Monthly Reporting

The specification of the Monthly Report was started by identifying the implementation approaches against to the requirements (TABLE II.) and by emphasising the needs of project members and managers in globally distributed projects. The selected approaches for monthly reporting at ABB are presented in the following table (TABLE III.).

TABLE III. APPROACHES FOR MONTHLY REPORTING AT ABB

R#	***Approach for Monthly Reporting at ABB***
R1	Financial data will be manually fetched from the ERP system. There is possibility to automate this in the future.
R2	We decided that drop down boxes contain default values, text boxes use the content from the previous month, and the project name is filled automatically. The project information is entered in the project site, but transferred to the project management department site for viewing, which allows everybody to use the site they are most familiar with.
R3	As the content of the report is available in the collaboration portal, everybody working in the unit can monitor the progress of the projects. Also, the line managers who don't have time to attend to the Follow-up meeting can use it as a way to keep up.
R4	Decision-making is streamlined, as the projects fill the Monthly Project Follow-Up beforehand, and they also explicitly state the required decisions, and therefore all the relevant line managers can prepare themselves for making the required decisions.
R5	Previously, the unit was very focused on monitoring the financial status of the projects only, instead, the new reporting focused on multisided view of the projects.

The project report (Monthly Project Report) was implemented containing at least the following issues to be controlled and documented monthly via the portal:

- Time Schedule (agreed with customer) and Comment on time schedule
- Financials (alternatives: On budget/Risk of exceeding budget /Budget exceeded)) and Comment on Financials
- Forecast updated in SAP
- Customer payments: (alternatives: No issues/Delays expected/Payments overdue)
- Resources & Comments on resources
- Technical risks/issues/Other risks
- Possible additional sales
- Positive/negative issues
- Main focus last month/main focus next month
- Proposal of needed decisions

Quality Assurance Reporting

The QA Report was also specified by identifying approaches against to the reporting requirements proposed (TABLE II.) and specific needs in global context. The approaches for the further specification were selected as introduced in the following table (TABLE IV.).

TABLE IV. APPROACHES FOR QA REPORTING AT ABB

R#	***Approach for QA Reporting at ABB***
R1	The report will be monitored in the follow-up meeting. Therefore, it is integrated to the process itself manually. There is possibility to improve this in the future.
R2	We decided that the project name is filled automatically. The project information is entered in the project site, but transferred to the project management department site for viewing, which allows everybody to use the site they are most familiar with.
R3	The content of the report is used to assure that the most important practices from the defined processes are in use. Also the KPIs collected based on the filled content are used to evaluate the overall situation of the challenge of "process are not followed".
R4	Controlling is streamlined, as summarized data is available for the decision makers. This was not the case earlier, and therefore controlling was time consuming activity.
R5	Earlier, the unit was very focused on monitoring the financial status of the projects only, the new reporting focused on multisided view of the projects.

IV. EXPERIENECES AND RESULTS OBTAINED

At first, the both reporting practices - the Monthly Project Report and QA Report - were taken into use as "hands-off" practice, which means that during the first follow-up meeting, the project managers reported the status verbally, and the status was entered to the portal during the meeting. This was to ensure an easy way of learning, and to avoid the need for a separate training session.

A. Experiences of Implementation

Experience of using SameRoomSpirit Wiki showed that the responsible people felt that most of the proposals presented were relevant even to organization that is not purely focused on SW development. The experiences also pointed out that the requirement to integrate processes and systems to obtain data in real time was the most difficult requirement to be implemented, and the integration was only done on the process level but not in the information systems. Therefore, there are possibilities to improve the functionality of the Monthly Follow-Up Report and the QA Report regarding the integrations.

For the Monthly Report, one could consider that the time schedule related to the information in the ERP system would be used to select the default value from the list of options for the time schedule related question. The same way the cost related information could be fetched from the ERP system (if the project is on budget and if there are issues with the customer payments). However, the free text comments should always be added by the project managers, and therefore fully automating the reporting based on data integration is not possible. Due to this fact, the value and the cost of system integration should be carefully evaluated.

For the QA Report, one could consider that all the planned dates and actual dates could be fetched from the ERP system. Alternatively, such report could be produced by ERP system or another reporting system, and the results would be visualized in the collaboration portal. Therefore, full integration for the QA Report could be the next step, if it is seen to bring value for the money. However, if the report should be used in distributed organization setup, where different participating units have different ERP systems or ways to implement, the integration would need to be done for each of the organizations separately.

B. Business Influence

As planned, the survey was repeated after the actions (Monthly Follow-Up Report and the QA Report) were executed for verifying the effectiveness of the improvement actions. During the spring 2010, it was measured that unnecessary or free work was reduced to almost half (45%): in the first survey (2009) unnecessary or free work was 19% of the project time whereas one year later, in the second survey, 10.5% of the project time was wasted to unnecessary or free work. It can't be claimed that the improvement was not only due to the implemented reporting practices because there were made some other improvement actions in the unit at the same time. Those actions cover general process and tool improvement actions that are continuously done in all big organisations, including, e.g., new and enhancement process instructions and templates and some new features implemented in engineering tools. The main effort and attention was paid in improving coordination and monitoring practices between and within the project by developing reporting practices as agreed based on the current state analysis.

Also, another main goal was successfully achieved: the visibility of the status of the customer projects had increased remarkably. Although reporting efforts of the project manager seemed to increase, it also provided them with a communication channel to inform the business management where their efforts were needed the most.

A positive signal was also that project monthly follow-up meetings had been kept regularly each month and reports had been filled in by the project managers into the Project Monthly Follow-Up tool. Especially the employees located outside the main office in distributed offices (at sites or in other engineering centers), appreciated the possibility to get information on all ongoing projects whenever they need the information. The other group benefiting from the follow-ups was people who were not tightly integrated into the projects, e.g., persons from purchasing, controllers, process development etc. Instead, the QA reporting usage was still low, proven that more improvement actions were needed in that process.

As sudden travels sometimes prevented project managers to join the meetings, back-up person to join the meeting was needed. Also, participation from sales to the project follow-ups was not as frequent as wished. On the other hand, the meetings provided good feedback on the improvement possibilities to sales as well. Therefore, it was suggested that improvement ideas were started to be entered into the collaboration portal, and published after the meeting to the relevant people.

The importance of QA reporting was recognized, however, it was unclear how to take the report fully into use at the moment. It had been considered that the management had a key role for committing new practices and tools. However, more training and information sharing was needed, too. There were also available some new features and tools for the portal that could be studied in the future.

V. CONCLUSIONS

The successful project quality control is a diverse and large entity. In this paper, the project controlling and monitoring activities were discussed from the project managers and the line management viewpoints in global context. The paper presented a successful case conducted at ABB focused on creating and improving project monitoring, controlling, and reporting practices to attain better coordination between and within the distributed projects as well as to build organization wide awareness of the status of the project portfolio.

The case study showed that the literature and industrial solutions presented in SameRoomSpirit Wiki were relevant even to organization that was not purely focused on SW development. The results also pointed out that project follow-up and controlling activities improve transparency, e.g., of the project status, between and within the projects through the whole organization. One of the major findings of the case was that unnecessary or free work had reduced even 45%. This improvement was largely due to the implemented reporting practices. Positive finding was also that project monthly follow-up meetings had been kept regularly each month and reports have been filled in by the project managers into the Project Monthly Follow-Up tool. Additionally, the visibility of the status of the customer projects had increased remarkably. The reporting efforts of the project manager increased during the case, and provided them with a communication channel to inform the business management where their efforts were needed the most.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024. The authors would like to thank the support of ITEA [15] and Tekes (the Finnish Funding Agency for Technology and Innovation) [16].

REFERENCES

[1] J. Hyysalo, P. Parviainen and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.

[2] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.

[3] J. S. Chou and W. K. Chong, "A web-based framework of project performance and control system," IEEE Conference on Robotics, Automation and Mechatronics (RAM 2008), IEEE, 2008, pp. 803-807.

[4] PRISMA, Productivity in Collaborative Systems Development, PRISMA project (2008-2011) homepage, URL: http://www.prisma-itea.org/ (Accessed 1.2.2011).

[5] SameRoomSpirit Wiki, Available only for registered users, URL: http://www.sameroomspirit.org (Accessed 1.2.2011).

[6] T. Ohno, The Toyota Production System: Beyond Large-Scale Production. Productivity Press, 1988.

[7] M. Bass and D. Paulish, "Global software development process research at siemens," The 3rd International Workshop on Global Software Development, in Proceedings of the International Conference on Software Engineering (ICSE 2004), 2004, pp. 11-14.

[8] D. Boland and B. Fitzgerald, "Transitioning from a co-located to a globally-distributed software development team: A case study at analog devices inc," The 3rd International Workshop on Global Software Development, in Proceedings of the International Conference on Software Engineering (ICSE 2004), 2004, pp. 4-7.

[9] J. D. Herbsleb and D. Moitra, "Global software development," Software, IEEE, vol. 18, (2), 2002, pp. 16-20.

[10] M. Paasivaara, "Communication needs, practices and supporting structures in global inter-organizational software development projects," The International Workshop on Global Software Development, in Proceedings of the International Conference on Software Engineering (ICSE 2003), 2003, pp. 59-63.

[11] R. D. Battin, R. Crocker, J. Kreidler and K. Subramanian, "Leveraging resources in global software development," Software, IEEE, vol. 18, (2), 2002, pp. 70-77.

[12] M. Paasivaara and C. Lassenius, "Collaboration practices in global inter-organizational software development projects," Software Process: Improvement and Practice, vol. 8, (4), 2003, pp. 183-199.

[13] C. Campana, E. Schott, M. Lappe and S. Haffner. Project portfolio management multi-project monitoring and reporting. The Campana & Schott Group.

[14] CMMI Product Team. (2010, Capability maturity model integration for development (CMMI-DEV, V1. 3). Carnegie Mellon.

[15] ITEA 2, Information Technology for European Advancement, ITEA 2 homepage, URL: http://www.itea2.org/ (Accessed 1.2.2011).

[16] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: http://www.tekes.fi/eng/ (Accessed 1.2.2011).

PAPER IV

Knowledge-related challenges and solutions in GSD

In: Expert Systems –
The Journal of Knowledge Engineering. 22 p.

Expert Systems
The Journal of Knowledge Engineering

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

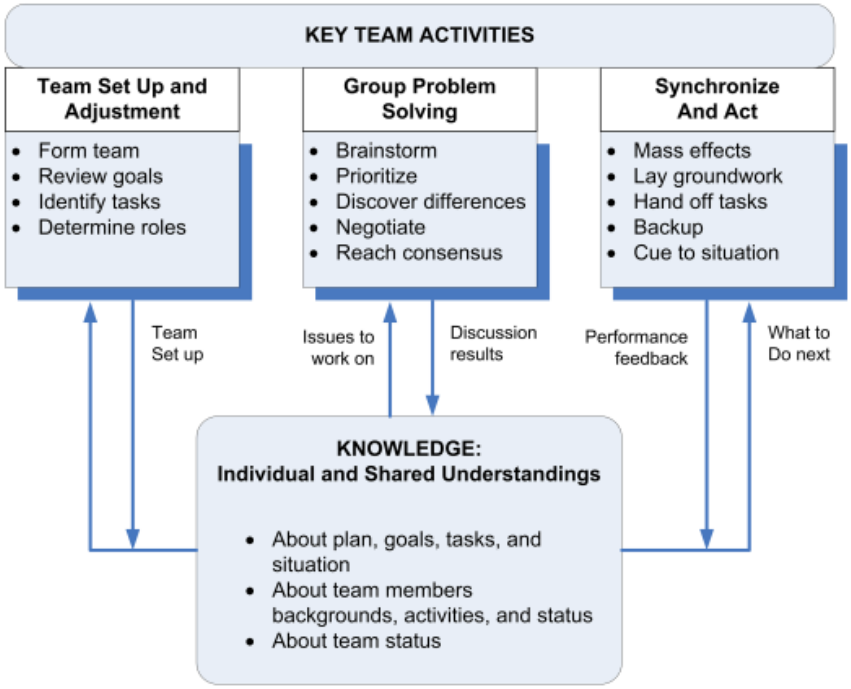

Figure 1: *The relationship between Knowledge and Team Activities (Noble, 2004).*

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

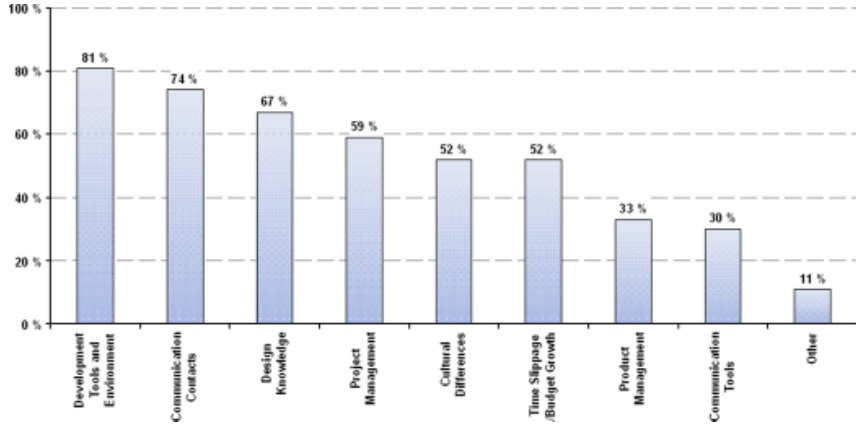

Figure 2: *Problem areas in distributed SW development (Komi-Sirviö & Tihinen, 2005).*

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

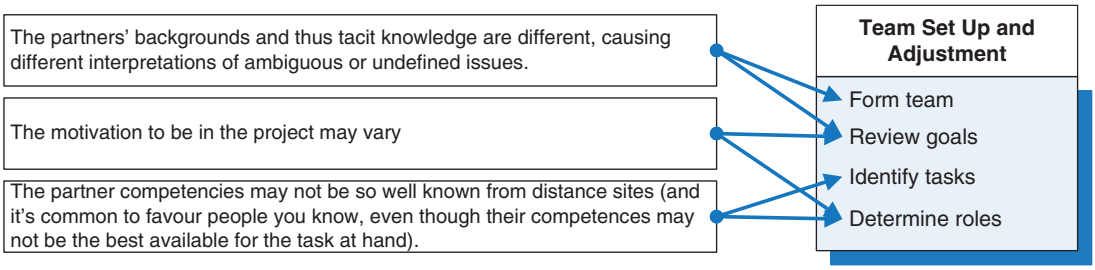

Figure 3: *Main knowledge based factors for 'Team Set Up and Adjustment'.*

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	

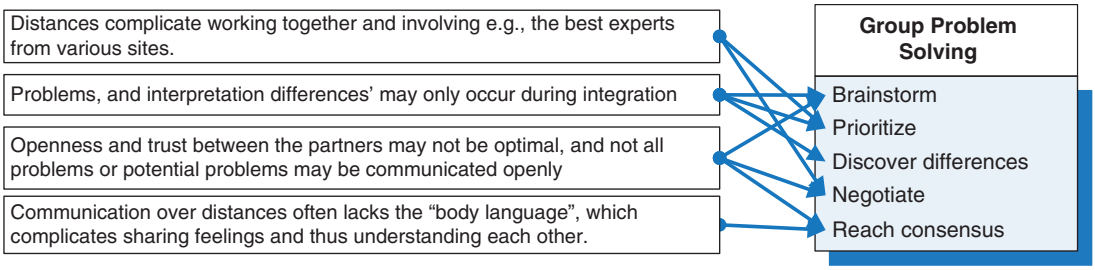

Figure 4: *Main knowledge based factors for 'Group Problem Solving'.*

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

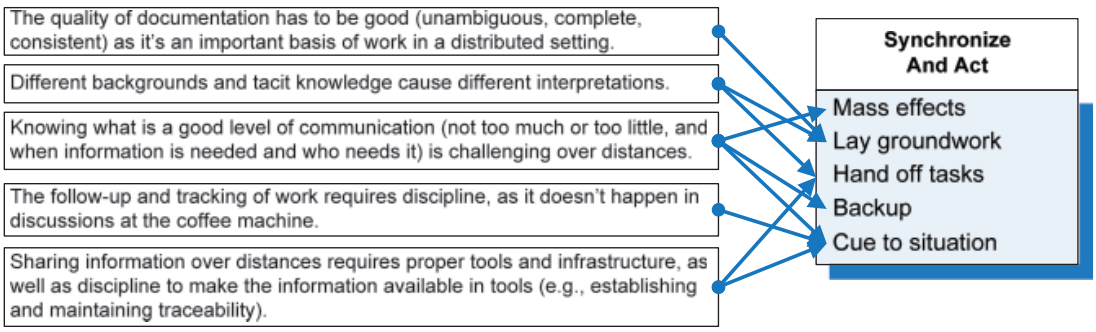

Figure 5: *Main knowledge based factors for 'Synchronize and Act'.*

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

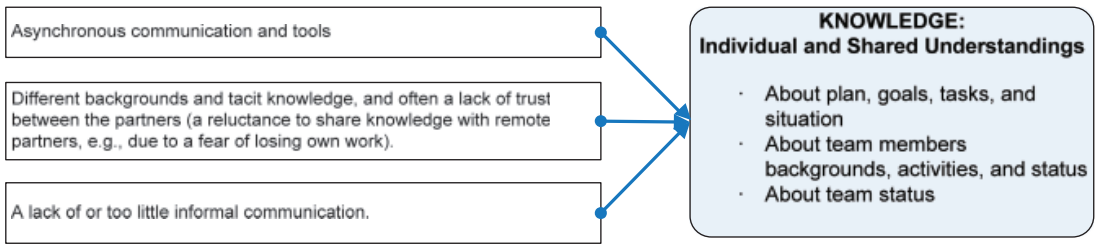

Figure 6: *Main knowledge based factors for 'Individual and Shared Understanding'.*

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

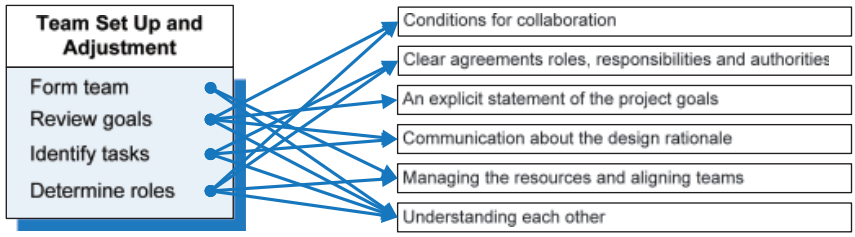

Figure 7: *Solutions relation to 'Team Set Up-up and Adjustment' activities.*

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

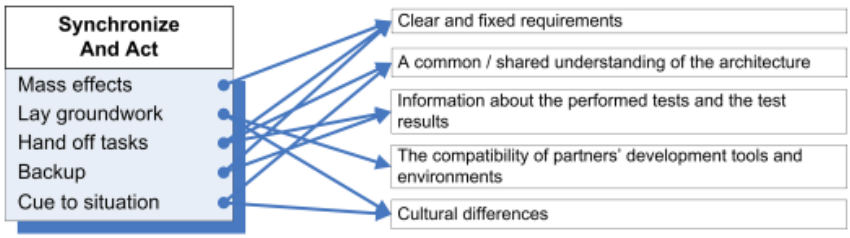

Figure 9: *Solutions relation to 'Synchronize and Act' activities.*

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

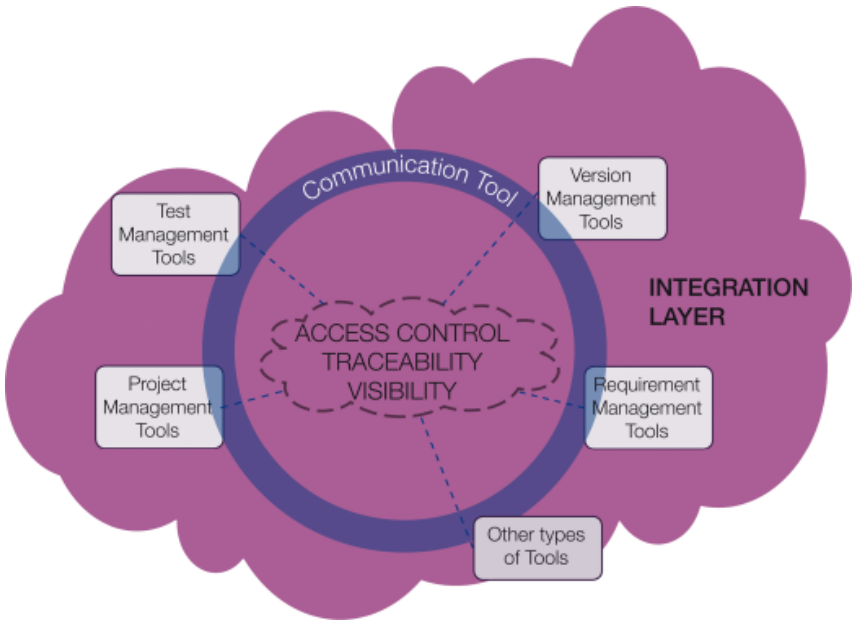

Figure 10: *Tool integration concepts.*

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

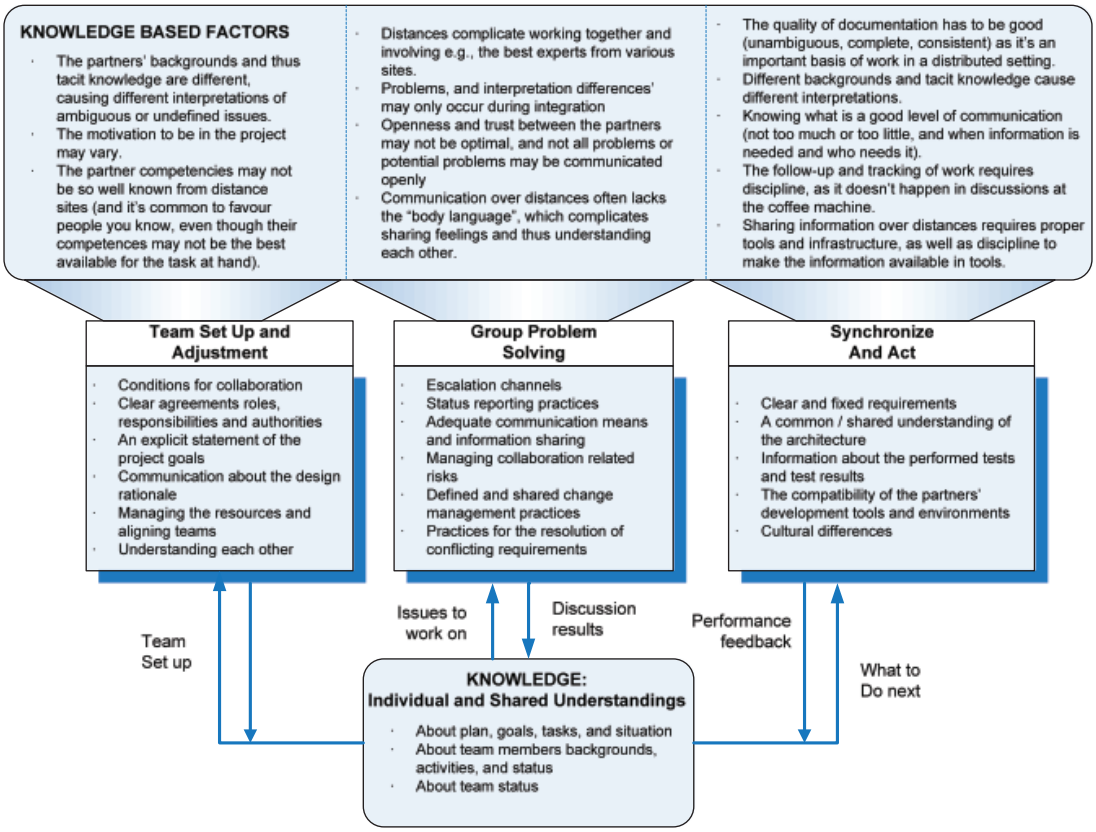

Figure 11: *Knowledge based factors relation to the team activities and solutions.*

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

Metrics in distributed product development

In: Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011). Barcelona, Spain.
Pp. 275–280.

Metrics in Distributed Product Development

Maarit Tihinen and Päivi Parviainen
Software Technologies
VTT Technical Research Centre of Finland
maarit.tihinen@vtt.fi
paivi.parviainen@vtt.fi

Rob Kommeren
Digital Systems & Technology
Philips,
The Netherlands
r.c.kommeren@philips.com

Jim Rotherham
Project Management Office
Symbio,
Finland
jim.rotherham@symbio.com

Abstract— **Nowadays the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers. However, management of a distributed product development project is proven to be more challenging and complicated than traditional one-site development. From the viewpoint of project management, the measurements and metrics are important activities for successful product development. This paper is focused on describing a set of metrics that is successfully used in industrial practice in distributed product development. Based on the experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed on a regular interval. One of the most important reasons for choosing these metrics was that they were aimed especially to provide early warning signals, i.e., means to proactively react to potential issues in the project. This is especially important in distributed projects, where specific means to track project status are needed.**

Keywords-metrics; measurements; global software development; distributed product development

I. INTRODUCTION

Globally distributed software development enables product development to take place independently of the geographical location of the individuals or organizations. In fact, nowadays the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers [1]. In practice distributed projects struggle with the same problems than single-site projects including problems related to managing quality, schedule and cost. Distribution only makes it even harder to handle and control these problems [2][3][4][5]. These challenges are caused by various issues, for example, less communication – especially informal communication – caused by distance between partners, and differences in background knowledge of the partners. That's why, in distributed projects the systematic monitoring and reporting of the project work is especially important, and measurement and metrics are an important means to do that effectively.

Management of a distributed product development project is more challenging than traditional development [6]. Based on an industrial survey [7], one of the most important topics in the project management in distributed software development is detailed project planning and control during the project. In global software development (GSD), this includes, e.g., dividing work by sites into sub-projects, clearly defined responsibilities, dependencies and timetables, along with regular meetings and status monitoring.

The main purpose of measurements and metrics in software production is to create means for monitoring and controlling and this way to provide support for decision making [8]. Traditionally, the software metrics are divided into process, product and resource metrics [9]. In the comprehensive measurement program, all these dimensions should be taken into consideration while interpreting measurement results, otherwise, the interpretation may lead to wrong decisions or incorrect actions. Successful measurement program can prove to be an effective tool for keeping on top of development effort, especially, for large distributed projects [10]. However, many problems and challenges have been identified that reduce and may even eliminate all interests to the measurements. For example, not enough time is allocated for measuring and metrics during a project, or not enough benefit is visibly gained by the project doing the measurement work (e.g., data is useful only at the end of project, not during the project). In addition, the "metric enthusiasts" may define too many metrics making it too time consuming. Thus, it's beneficial [10] to define core metrics to collect across all projects to provide benchmarking data for projects, and to build on measures that come naturally out of existing processes and tools.

This paper is focused on describing a metrics set that are successfully used in distributed product development. The main purpose of the paper is to offer a set of essential metrics with experiences of their use. The amount of the metrics is knowingly kept as limited as possible. Also, the metrics should be such, that they provide online information during the projects, in order to enable fast reaction to potential problems during the project. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. Royal Philips Electronics is a global company providing healthcare, consumer life-style

 ISBN: 978-1-61208-165-6

and lighting products and services. Digital Systems & Technology is a unit within Philips Research that develops first of a kind products in the area of healthcare, well-being and lifestyle. The projects follow a defined process and are usually distributed over sites and/or use subcontractors as part of product development. Symbio Services Oy provides tailored services to organizations seeking to build tomorrow's technologies. Well-versed in a variety of software development methodologies and testing best practices, Symbio's specialized approaches and proprietary processes begin with product design and stem through globalization, maintenance and support. Symbio has built a team of worldwide specialists that focus on critical areas of the product development lifecycle. Currently Symbio employs around 1400 people and their project execution is distributed between sites in the US, Sweden, Finland and China.

The paper is structured as follows. Firstly, an overview of related work – literature studies and their limitations related to measurements and metrics of distributed product development – is introduced in Section II. Then, proposed metrics are presented using Rational Unified Process (RUP) [11] approach as a framework. After that, industrial experiences of using the metrics are discussed. Finally, the conclusions are drawn in Section V.

II. MEASUREMENTS IN GSD

There are several papers that discuss globally distributed software engineering and its challenges, for example, [10], [12] and [13]. Also, metrics in general and for specific aspects have been discussed in numerous papers and books for decades. However, little global software development (GSD) literature has focused on metrics and measurements or even discusses the topic. Da Silva et al. [6] report similar conclusion based on analysis of DSD literature published during 1999 – 2009: they state as one of their key finding that the "vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of distributed software development (DSD) project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature."

The papers that have discussed some metrics for GSD usually focus on some specific aspect, for example, Korhonen and Salo [13], discuss quality metrics to support defect management process in a multi-site organization. Simmons and Ma [14] discuss a software engineering expert system (SEES) tool where the software professional can gather metrics from CASE tool databases to reconstruct all activities in a software project from project initiation to project termination. Misra [15] presents a cognitive weight complexity metric (CWCM) for unit testing in a global software development environment. Lotlikar et al. [16] propose a framework for global project management and governance including some metrics with main aim to support work allocation to various sites. Peixoto et al. [12] discuss effort estimation in global software development, and one of their conclusions is that "GSD projects are using all kinds of estimation techniques and none of them is being consider as proper to be used in all cases that it has been used", meaning, that there is no established technique for GSD projects.

Some effort has also been invested in defining how to measure success of GSD projects [17], and these metrics mainly focus on cost related metrics and are done after project completion. The focus of this paper is to discuss metrics for monitoring ongoing GSD projects and that way identify needs for corrective actions early.

A. Traditional metrics and project characteristics

Software measurements and metrics have been discussed since 1960's. The metrics have been classified many different ways, for example, they can be divided into basic and additional metrics [18] where basic metrics are size, effort, schedule and defects, and the additional metrics are typically metrics that are calculated or annexed from basic metrics (e.g., productivity = software size per used effort). The metrics can be divided also into objective or subjective metrics [18]. The objective metrics are easily quantified and measured, examples including size and effort, while the subjective metrics include less quantifiable data such as quality attitudes (e.g., excellent, good, fair, poor). An example of the subjective metrics is customer satisfaction. Furthermore, software metrics can be classified according to the entities of product, processes and resources [9]. Example metrics of product entities are size, complexity, reusability and maintainability. Example metrics of process entities are effort, time, number of requirements changes, number of specification/coding faults found and cost. Furthermore, examples of resource entities are age, price, size, maturity, standardization certification, memory size or reliability. These classifications, various viewpoints and the amount of examples merely prove how difficult the selection of metrics really can be during the project.

In addition to different ways of metrics classification, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparison should be done within the same kind of development projects, or the differences should be taken into account. Traditional project characteristics are, e.g., size and duration of a project, type of a project (development, maintenance, operational lifetime etc.), project position (contractor, subcontractor, internal development etc.), type of software (hardware-related software development, application software, etc.) or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model etc.). Furthermore, different phases of development projects have to be taken consideration while analyzing gathered measurement data.

B. Metrics and measurements during product development

A phase of lifecycle of development project affects to the interpretation of the metrics. Thus, in this paper, proposed metrics are introduced by using commonly known approach of software development Rational Unified Process (RUP). RUP is a process that provides a disciplined approach to

 ISBN: 978-1-61208-165-6

assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget [11].

The software lifecycle is divided into cycles, each cycle working on a new generation of the product. RUP divides one development cycle in four consecutive phases [11]: 1) inception phase, 2) elaboration phase, 3) construction phase and 4) transition phase. Furthermore, there can be one or more iterations within each phase during the software generation. The phases and iterations of RUP approach are illustrated in following Figure 1.

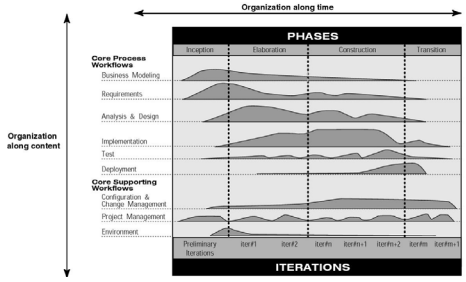

Figure 1. Phases and Iterations of RUP approach [11].

From a technical perspective the software development is seen as a succession of iterations, through which the software under development evolves incrementally [11]. From measurement perspective this means that some metrics can be focused on one or two phases of the development cycle, and some can be continuous metrics that can be measured in all phases, and can be analysed, e.g., in iterations.

C. Measurements and metrics in GSD

Software measurement is defined by [19] as follows: "The software measurements is the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products". In the daily software development work, the measurements are still seen as unfamiliar or even an extra burden for projects. For example, project managers feel it as time consuming to collect metrics for the organization (e.g., business-goal-related metrics) while they need to have metrics that are relevant to the project. Furthermore, they have impressed that there has not been budgeted enough time for measurements, and that's why it's really difficult to get approval from stakeholders for this kind of work [10].

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools or their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. In addition, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that their comparison is impossible. Thus, the interpretation of measurements data is more complicated in GSD than one site projects. That's why it's recommended to select moderate amount of metrics. In this paper we will present a set of metrics to use during GSD. Also industrial experiences about the metrics will be discussed.

The common metrics (effort, size, schedule etc.) are also applicable for GSD projects. However, special attention may be needed in training the metrics collection, to ensure common understanding of them (e.g., used classifications). Also, as measurements also tend to guide people's behavior, it's important to ensure that all are aware of the purpose of the metrics (i.e., not to measure individual performance), specifically in projects distributed over different cultures.

III. EXAMPLES OF INDUSTRIAL PRACTICES

In this Section the metric set used in the companies is introduced. The metrics are introduced according to the RUP phases where the metric is seen most relevant to measure. For each metric, a name, a notation and a detailed definition is introduced. The main goal is to offer a useful, yet a reasonable amount of metrics, for supporting the on-time monitoring of the GSD projects. Thus, the indicators are supposed to be leading indicators rather than lagging indicators, for example, planned / actual schedule measurements should be implemented as milestone trend analysis: measure the slip in the first milestone and predict the consequences for the other milestones and project end.

A. Metrics for Inception Phase

During the inception phase, the project scope has to be defined and the business case has to be established. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones. Inception is the smallest phase in the project, and ideally it should be quite short. Example outcomes of the inception phase are a general vision document of the core project's requirements, main constraints, an initial use-case model (10% -20% complete), and a project plan, showing phases and iterations [20]. Proposed metrics to be taken consideration in this phase are introduced in Table I.

TABLE I. METRICS FOR THE INCEPTION PHASE

Metric	Notation	Definition
Planned Schedule	$D_{PLANNED}$	The planned Date of delivery (usually the completion of an iteration, a release or a phase)
Planned Personnel	# $FT_{PLANNED}$	The planned number of Full Time persons in the project at any given time
Proposed Requirements	# Reqs	The number of proposed requirements.

The metrics Planned Schedule and Planned Personnel are mostly needed for comparison with actual schedule and

 ISBN: 978-1-61208-165-6

personnel, in order to identify lack of available resources as well as delays in schedule quickly. The amount of Proposed Requirements tells about the progress of the product definition.

B. Metrics for Elaboration Phase

During the elaboration phase a majority of the system requirements is expected to capture. The purpose of the phase is to analyze the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the construction phase. Example outcomes of the elaboration phase are a use-case model (at least 80% complete), a software architecture description, supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case, a revised risk list and a revised business case, and a development plan for the overall project. Proposed metrics to be taken consideration in this phase are introduced in Table II.

TABLE II. METRICS FOR THE ELABORATION PHASE

Metric	Notation	Definition
Schedule: Planned /Actual Schedule	 $\mathbf{D}_{\text{PLANNED}}$ $\mathbf{D}_{\text{ACTUAL}}$	The planned/actual Date of delivery (usually the completion of an iteration, a release or a phase)
Staff: Planned /Actual Personnel	 $\mathbf{\#FT}_{\text{PLANNED}}$ $\mathbf{\#FT}_{\text{ACTUAL}}$	The planned/actual number of Full Time persons in the project at any given time
Requirements -Proposed -Accepted -Not implemented	 $\mathbf{\#Reqs}_{\text{PROP.}}$ $\mathbf{\#Reqs}_{\text{ACCEP.}}$ $\mathbf{\#Reqs}_{\text{NOT\_IMPL.}}$	The number (#) of - proposed requirements - reqs accepted by customer - not implemented reqs
Tests -Planned	 $\mathbf{\#Tests}_{\text{PLANNED}}$	The number (#) of - planned tests
Documents: -Planned -Proposed -Accepted	 $\mathbf{\#Docs}_{\text{PLANNED}}$ $\mathbf{\#Docs}_{\text{PROPOSED}}$ $\mathbf{\#Docs}_{\text{ACCEPTED}}$	The number (#) of planned /proposed /accepted documents to be reviewed during the project.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Staffing metric may explain deviations in the expected progress vs. the actual progress, both from technical as well as from schedule viewpoint. Note that those metrics that are more relevant to measure by iterations (e.g., effort and size) are introduced later (in Section E).

C. Metrics for Construction Phase

Construction is the largest phase in the project. During the phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. Example outcomes of the phase consist of a software product integrated on the adequate platforms, user manuals, and a description of the current release. Proposed metrics to be taken consideration in this phase are introduced in Table III. Note that those metrics that are continuously measured are introduced later (in Section E).

TABLE III. METRICS FOR THE CONSTRUCTION PHASE

Metric	Notation	Definition
Planned /Actual Schedule Planned /Actual Personnel	$\mathbf{D}_{\text{PLANNED}}$ $\mathbf{D}_{\text{ACTUAL}}$ $\mathbf{\#FT}_{\text{PLANNED}}$ $\mathbf{\#FT}_{\text{ACTUAL}}$	Defined in the elaboration phase.
Requirements: -Proposed -Accepted -Not implemented -Started -Completed	 $\mathbf{\#Reqs}_{\text{PROP.}}$ $\mathbf{\#Reqs}_{\text{ACCEP.}}$ $\mathbf{\#Reqs}_{\text{NOT\_IMPL.}}$ $\mathbf{\#Reqs}_{\text{STARTED}}$ $\mathbf{\#Reqs}_{\text{COMPLETED}}$	The number (#) of - proposed requirements - reqs accepted by customer - not implemented reqs - reqs started to implement - reqs completed
Change Requests: -New CR -Accepted -Implemented	 $\mathbf{\#CRs}_{\text{NEW}}$ $\mathbf{\#CRs}_{\text{ACCEPTED}}$ $\mathbf{\#CRs}_{\text{IMPL.}}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented
Tests: -Planned -Passed -Failed -Not tested	 $\mathbf{\#Tests}_{\text{PLANNED}}$ $\mathbf{\#Tests}_{\text{PASSED}}$ $\mathbf{\#Tests}_{\text{FAILED}}$ $\mathbf{\#Tests}_{\text{NOT TESTED}}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
Defects -by Priority: e.g., Showstopper, Medium, Low	 $\mathbf{\#Dfs}_{\text{PRIORITY}}$	The number (#) of - defects by Priority during the time period
Documents: -Planned -Proposed -Accepted	 $\mathbf{\#Docs}_{\text{PLANNED}}$ $\mathbf{\#Docs}_{\text{PROPOSED}}$ $\mathbf{\#Docs}_{\text{ACCEPTED}}$	Defined in the elaboration phase.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both on the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Defect metrics tell both of the progress of testing, as well as maturity of the product.

D. Metrics for Transition Phase

The final project phase of the RUP approach is transition. The purpose of the phase is to transfer a software product to a user community. Feedback received from initial release(s) may result in further refinements to be incorporated over the course of several transition phase iterations. The phase also includes system conversions, installation, technical support, user training and maintenance. From measurements viewpoint the metrics identified in the phases relating to schedule, effort, tests, defects, change requests and costs are still relevant in the transition phase. In addition, customer satisfaction is generally gathered in the transition phase.

E. Metrics for Iterations

Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release. Each release is accompanied by supporting artifacts: release description, user's documentation, plans, etc. Although most iterations will include work in most of the process disciplines (e.g.,

 ISBN: 978-1-61208-165-6

requirements, design, implementation, testing) the relative effort and emphasis will change over the course of the project. Proposed metrics to be taken consideration in this phase are introduced in Table IV.

TABLE IV. METRICS FOR ITERATIONS

Metric	Notation	Definition
Effort: -Planned Effort -Actual Effort	$E_{PLANNED}$ E_{ACTUAL}	The planned/actual effort required of any given iteration of the project.
Size: -Planned size -Actual size	$SIZE_{PLANNED}$ $SIZE_{ACTUAL}$	The planned /actual size of each iteration can be measured as SLOC (Source Lines of Code), Points or any other commonly accepted way.
Cost: -Budgeted -Expenditure	$COST_{BUDGET}$ $COST_{ACTUAL}$	The budgeted cost /actual expenditure for any given iteration.
Velosity: -planned /actual story points	$\#PTS_{PLAN}$ $\#PTS_{ACT}$	How many story points are planned to be /actually implemented of any given iteration of the project.
Productivity:	$\frac{E_{ACTUAL}}{\#PTS_{ACTUAL}}$	Use effort per acutally implemented story points for each sprint /iteration

All of these metrics provide indication of the project progress and reasons for deviations should be analysed. These metrics should be analysed together with other metrics results (presented in Tables I-III) in order to gain comprehensive picture of the status.

IV. EXPERIENCES AND DISCUSSION

The metrics presented in previous section were common for both of the companies. Although the metrics were chosen independently by both companies, the reasoning behind choosing these metrics was similar. An important reason was to come from a re-active into a pro-active mode, i.e., to introduce 'early warning' signals for the project and management. Specifically these metrics have been chosen as they indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort, distributed over sites and companies. Accordingly, the metrics set can be seen as a 'balanced score card', on which management can take the right measures, balancing insights from time, effort (e.g., staffing), cost, functionality (requirements) and quality (tests) perspective.

An important aspect was also that the metrics are easy of capture and that they can be captured from the used tools "for free", or can be quickly calculated at regular intervals. Costs and budgets are good examples of metrics that can be easily captured from the tools. This is also important from GSD viewpoint, as automated capturing reduces the chance of variations caused by differences in recording the metrics data in different sites. Neither of the companies use metrics based on lines-of-code as they did not find it to be a reliable indicator of progress, size or quality of design.

As can be seen, the metrics are quite similar as in single site development. However, the metrics may be analysed separately for each site, and comparisons between sites can thus be made in order to identify potential problems early. Also, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires 'super-balancing': how to come to the right corrective action if for instance, in one side the % of not accepted requirements is high, and in the other side the # of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, relating to subjective metrics, such as effort estimation, differences between backgrounds of the people (e.g., cultural or work experience) in different sites may affect the result.

The companies also use the measurement results to gain insight into why a measure varies between similar single site and multi-site projects in order to try to reduce potential variances. This also partially explains the use of the same metrics as single-site development. Furthermore, the challenges in communication and dynamics of distributed teams mean that working practices need to be addressed continuously. However, in addition to metrics results, paying close attention and acting on feedback from retrospectives is as important, if not more important than drawing strong conclusions from metrics alone.

Currently, both companies are in process of revamping their metric usage, but feel confident that these metrics are the right ones. Easy implementation and by that easy acceptance is the most crucial thing to get these metrics as established practice within the company.

Both companies are careful in introducing new metrics, as it's well known that too many metrics leads to overkill and rejection by the organization, and does not provide the right insights and indication for control measures. However, a potential measurement to be added to the set specifically from distributed development viewpoint, could be measurements related to time spent idling, i.e., waiting for something, and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when a team is not performing. These additional metrics should be focused on measuring the project performance, especially task and team performance in GSD.

V. CONCLUSION

The management of the increasingly common distributed product development project is proven to be more challenging and complicated than traditional one-site development. Metrics are seen as important activities for successful product development as they provide means to effectively monitor the project progress. However, defining useful, yet reasonable amount of metrics is challenging, and

 ISBN: 978-1-61208-165-6

there is little guidance available for a company to define metrics for its distributed projects.

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools or their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. Furthermore, especially interpretation and decision-making based on the measurement results require that the distributed development implications are taken carefully into consideration.

This paper focused on describing a set of metrics that is successfully used in industrial practice in distributed product development. These metrics, are aimed especially to provide means to proactively react to potential issues in the project, and are meant to be used as a whole, not interpreted as single information of project status.

The metrics presented in the paper were common for both of the companies. Based on experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed at regular interval. Also, one of the most important reasons was that these metrics were aimed especially to provide means to proactively react to potential issues in the project. The balancing insights from time, effort, cost, functionality and quality was also seen as very important aspect.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024 [21]. The authors would like to thank the support of ITEA [22] and Tekes (the Finnish Funding Agency for Technology and Innovation) [23].

REFERENCES

[1] J. Hyysalo, P. Parviainen, and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.

[2] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," In Proceedings of Future of Software Engineering FOSE '07, IEEE Computer Society, 2007, pp. 188-198.

[3] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, 2000, pp. 319-328.

[4] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," Advances in Software Engineering, 2009, pp. 14.

[5] S. Komi-Sirviö and M. Tihinen, "Lessons learned by participants of distributed software development," Knowledge and Process Management, vol. 12, (2), 2005, pp. 108-122.

[6] F. Q. B. da Silva, C. Costa, A. C. C. França, and R. Prikladinicki, "Challenges and solutions in distributed software development project management: A systematic literature review," In Proceedings of International Conference on Global Software Engineering (ICGSE2010), IEEE, 2010, pp. 87-96.

[7] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.

[8] V. R. Basili, "Software modeling and measurement: The Goal/Question/Metric paradigm," 1992.

[9] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co. Boston, MA, USA, 1998.

[10] M. Umarji and F. Shull, "Measuring developers: Aligning perspectives and other best practices," IEEE Software, vol. 26, (6), 2009, pp. 92-94.

[11] P. Kruchten, The Rational Unified Process: An Introduction. Addison-Wesley Professional, 2004.

[12] C. E. L. Peixoto, J. L. N. Audy, and R. Prikladnicki, "Effort estimation in global software development projects: Preliminary results from a survey," In Proceedings of International Conference on Global Software Engineering, IEEE Computer Society, 2010, pp. 123-127.

[13] K. Korhonen and O. Salo, "Exploring quality metrics to support defect management process in a multi-site organization - A case study," In Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2008, pp. 213-218.

[14] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," In Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), IEEE, 2006, pp. 33-38.

[15] S. Misra, "A metric for global software development environment," In Proceedings of the Indian National Science Academy 2009, pp. 145-158.

[16] R. M. Lotlikar, R. Polavarapu, S. Sharma, and B. Srivastava, "Towards effective project management across multiple projects with distributed performing centers," In Proceedings of IEEE International Conference on Services Computing (CSC'08), IEEE, 2008, pp. 33-40.

[17] B. Sengupta, S. Chandra, and V. Sinha, "A research agenda for distributed software development," In Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 731-740.

[18] K. H. Möller and D. J. Paulish, Software Metrics: A Practitioner's Guide to Improved Product Development. Institute of Electrical & Electronics Enginee, London, 1993.

[19] R. Van Solingen and E. Berghout, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill, 1999.

[20] P. Kruchten, "A rational development process," CrossTalk, vol. 9, (7), 1996, pp. 11-16.

[21] PRISMA, Productivity in Collaborative Systems Development, PRISMA project (2008-2011) homepage, URL: http://www.prisma-itea.org/ (Accessed 1.6.2011).

[22] ITEA 2, Information Technology for European Advancement, ITEA 2 homepage, URL: http://www.itea2.org/ (Accessed 1.6.2011).

[23] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: http://www.tekes.fi/eng/ (Accessed 1.6.2011).

 ISBN: 978-1-61208-165-6

PAPER VI

Metrics and measurements in global software development

In: International Journal on Advances in Software, Vol. 5, No. 3&4, pp. 278–292.

Metrics and Measurements in Global Software Development

Maarit Tihinen and Päivi Parviainen
Digital Service Research
VTT Technical Research Centre of Finland
maarit.tihinen@vtt.fi
paivi.parviainen@vtt.fi

Rob Kommeren
Digital Systems & Technology
Philips,
The Netherlands
r.c.kommeren@philips.com

Jim Rotherham
Project Management Office
Symbio,
Finland
jim.rotherham@symbio.com

Abstract—**Today products are increasingly developed globally in collaboration between subcontractors, third-party suppliers and in-house developers. However, management of a distributed product development project is proven to be more challenging and complicated than traditional single-site development. From the viewpoint of project management, the measurements and metrics are important elements for successful product development. This paper is focused on describing a set of essential metrics that are successfully used in Global Software Development (GSD). In addition, visualised examples are given demonstrating various industrial experiences of use. Even if most of the essential metrics are similar as in single-site development, their collection and interpretation need to take into account the GSD aspects. One of the most important reasons for choosing proposed metrics was their provision of early warning signs - to proactively react to potential issues in the project. This is especially important in distributed projects, where tracking the project status is needed and more complex. In this paper, the first ideas of GSD specific metrics are presented based on the common challenges in GSD practice.**

Keywords-metrics; measurements; global software development; distributed product development

I. INTRODUCTION

Global Software Development (GSD) is increasingly common practice in industry due to the expected benefits, such as lower costs and utilising resources globally. GSD brings several additional challenges to the development, which also affects the measurement practices, results and metrics interpretation. A current literature study showed that there is little research on GSD metrics or experiences of their use. This paper is enhanced and extended version of the ICSEA 2011 conference paper "Metrics in distributed product development" [1] where the metrics set had been successfully used in GSD were introduced. In this paper, the published metric set (with an example set of visualised metrics) was given with industrial experiences of their use. In addition, challenges faced during GSD are discussed from the viewpoint of metrics and measurements as well as potential GSD specific metrics.

Software metric is a valuable factor for the management and control of many software related activities, for example; cost, effort and schedule estimation, productivity, reliability and quality measures. Traditionally software measurement has been understood as an information gathering process. For example, software measurement is defined by [2] as follows: "The software measurements is the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products". The measurement data item consists of numeric data (e.g., efforts, schedules) or a pre-classified set of categories (e.g., severity of defects: minor, medium, major). Software metrics can consist of several measurement data items singly or in combination. Metric visualisation is a visual representation of the collected and processed information about software systems. Typically software metrics are visualised for presenting this information in a meaningful way that can be understood quickly. For example, visualising metrics through charts or graphs is usually easier to understand than long textual or numerical descriptions.

The main purpose of measurements and metrics in software production is to create the means for monitoring and controlling which provide support for decision-making and project management [3]. Traditionally, the software metrics are divided into process, product and resource metrics [4]. In the comprehensive measurement program, all these dimensions should be taken into consideration while interpreting measurement results; otherwise the interpretation may lead to wrong decisions or incorrect actions. A successful measurement program can prove to be an effective tool for keeping on top of the development effort, especially for large distributed projects [5]. However, many problems and challenges have been identified that reduce and may even eliminate all interests to the measurements. For example, not enough time is allocated for the measurement activities during a project, or not enough visible benefits are gained by the project doing the measurement work (e.g., data is useful only at the end of

project, not during the project). In addition, the "metric enthusiasts" may define too many metrics making it too time consuming to collect and analyse the data. Thus, it's beneficial [5] to define core metrics to collect across all projects to provide benchmarking data for projects, and to focus on measurements that come naturally out of existing practices and tools.

GDS development enables product development to take place independently of the geographical location, individuals or organizations. In fact, today the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers [6]. In practice distributed projects struggle with the same problems as single-site projects including problems related to managing quality, schedule and cost. Distribution only makes it even harder to handle and control these problems [7][8][9][10][11]. These challenges are caused by various issues, for example, less communication – especially informal communication – caused by distance between partners, and differences in background knowledge of the partners. That's why, in distributed projects the systematic monitoring and reporting of the project work is especially important, and measurement and metrics are an important means to do that effectively.

Management of a distributed product development project is more challenging than traditional development [12]. Based on an industrial survey [13], one of the most important topics in the project management in distributed software development is detailed project planning and control during the project. In GSD, this includes; dividing work by sites into sub-projects, clearly defined responsibilities, dependencies and timetables, along with regular meetings and status monitoring.

In this paper, a set of essential metrics used in GSD is discussed with experiences of their use. The main purpose is to introduce the selected metric set from the viewpoint of their proactive role in decision-making during globally distributed software development. The chosen metrics indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but requires special effort, distributed over sites and companies.

The amount of the metrics is intentionally kept as limited as possible. Also, the metrics should be such, that they provide online information during the projects, in order to enable fast reaction to potential problems during the project. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. Royal Philips Electronics is a global company providing healthcare, consumer lifestyle and lighting products and services. Digital Systems & Technology is a unit within Philips Research that develops first-of-a-kind products in the area of healthcare, well-being and lifestyle. The projects follow a defined process and are usually distributed over sites and/or use subcontractors as part of product development. Symbio Services Oy provides tailored services to organizations seeking to build tomorrow's technologies. Well-versed in a variety of software development methodologies and testing best practices, Symbio's specialized approaches and proprietary processes begin with product design and continue through globalization, maintenance and support. Symbio has built a team of worldwide specialists that focus on critical areas of the product development lifecycle. Currently, Symbio employs around 1400 people and their project execution is distributed between sites in the US, Sweden, Finland and China.

The metrics and discussion in the paper is based on GSD improvement work carried out during several years, in several research projects, including experiences from 54 industrial cases (see Parviainen [14], SameRoomSpirit Wiki [15]). This paper focuses especially on the experiences of two companies, Philips and Symbio.

The paper is structured as follows. Firstly, an overview of related work – available literature and its limitations related to measurements and metrics in distributed product development. This is introduced in Section II. In Section III, basic GSD circumstances with challenges are presented in order to explain the special requirements for measurements in GSD where the proposed metrics set is to be collected and utilised. In Section IV, measurement and metrics background and used terminology are introduced. In Section V, proposed metrics are presented using Rational Unified Process (RUP) [16] approach as a framework. The proposed metric set is presented with visualised examples and industrial experiences of their use. Furthermore, some GSD specific metrics are introduced in Section VI. Finally, discussion about metrics and their experiences is presented in Section VII and the conclusions are discussed in Section VIII.

II. RELATED WORK

There are several papers that discuss globally distributed software engineering and its challenges, for example, [5], [17] and [18]. Also, metrics in general and for specific aspects have been discussed in numerous papers and books for decades. However, little GSD literature has focused on metrics and measurements or even discusses the topic. Da Silva et al. [12] report similar conclusion based on analysis of distributed software development (DSD) literature published during 1999 – 2009: they state as one of their key findings that the "vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of DSD project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature." Bourgault et al. [19] reported similar findings, "Clearly, research into distributed projects' performance metrics and measurement needs more attention from researchers and practitioners so that it can contribute to the development and diffusion of well-designed management information systems."

The papers that have discussed some metrics for GSD usually focus on some specific aspect, for example, Korhonen and Salo [18], discuss quality metrics to support

defect management process in a multi-site organization. Misra [20] presents a cognitive weight complexity metric (CWCM) for unit testing in a global software development environment. Lotlikar et al. [21] propose a framework for global project management and governance including some metrics with the main goal to support work allocation to various sites. Lane and Agerfalk [22] use another framework as an analytic device to investigate various projects performed by distributed teams in order to explore further the mechanisms used in industry both to overcome obstacles posed by distance and process challenges and also to exploit potential benefits enabled by GDS. Similarly, Piri and Niinimäki [23] applied Word Design Questionnaire (WDQ) that consists of total of 21 sum variables in four categories (task characteristics, knowledge characteristics, social characteristics, and work context) to compare differences between the co-located and the distributed projects by metrics - "work design", "team dynamics", "teamwork quality", "project performance" and "individual satisfaction". These kinds of frameworks could be used to evaluate effectiveness of distributed team configuration during GSD projects as well. Peixoto et al. [17] discuss effort estimation in GSD, and one of their conclusions is that "GSD projects are using all kinds of estimation techniques and none of them is being consider as proper to be used in all cases that it has been used", meaning, that there is no established technique for GSD projects. In addition, some effort has also been invested in defining how to measure success of GSD projects [24], and these metrics mainly focus on cost related metrics and are done after project completion. These papers usually use common metrics that are not specific for GSD projects. For example, Ramasubbu and Balan [25] use 11 metrics (productivity, quality, dispersion, prevention QMA (Quality Management Approach), appraisal QMA, failure QMA, code size, team size, design rework, upfront investment and reuse), development productivity and conformance quality to evaluate how work dispersion effects to identified metrics. However, these metrics have not been used to gather information, indicators or experiences from ongoing distributed development.

Furthermore, only few papers discuss measurement tooling for GSD projects. Simmons [26] describes a PAMPA tool, where an intelligent agent tracks cost driver dominators to determine if a project may fail and tells managers how to modify project plans to reduce probability of project failure. Additionally, Simmons and Ma [27] discuss a software engineering expert system (SEES) tool where the software professional can gather metrics from CASE tool databases to reconstruct all activities in a software project from project initiation to project termination. Da Silva et al [28] discuss software cockpits from GSD viewpoint. They propose to examine various visualizations in the context of software cockpits, at-a-glance computer controlled displays of development-related data collected from multiple sources. They present three visualizations: (1) shows high-level information about teams and dependencies among them in an interactive world map, (2) displays the system design through a self-updating view of the current state of the software implementation, and (3) is a 3D visualization that presents an overview of current and past activities in individual workspaces.

The focus of this paper is to introduce a metrics set that creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. Furthermore, the paper gives several visualised examples of metrics that can be utilised while monitoring on-going GSD projects. The introduced metrics set can be seen as 'balanced score card', on which management can balance insights (~status) from time, effort, cost, functionality (requirements) and quality (tests) perspective.

III. BASIC GSD CIRCUMSTANCES WITH CHALLENGES

Parviainen [14] describes problems and challenges that are directly caused by the basic GSD circumstances. These challenges influence measurements and metrics and their interpretation during distributed software development. These challenges are mainly an intrinsic and natural part of GSD and they can either complicate globally distributed product development or even cause further challenges. The basic circumstances are:

- Multiple parties, meaning two or more different teams and sites (locations) of a company or different companies.
- Time difference and distance that are caused by the geographical distribution of the parties.

Problems caused by these circumstances include; issues such as unclear roles and responsibilities for the different stakeholders in different parties or locations, knowing the contact persons (e.g., responsibilities, authorities and knowledge) from different locations and establishing and ensuring a common understanding across distance. The basic GSD circumstances can also lead to poor transparency and control of remote activities as well as difficulties in managing dependencies over distance, problems in coordination and control of the distributed work and integration problems, for example. Problems may also be caused by basic circumstances in terms of accessing remote databases and tools or accordingly 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. The basic circumstances may also cause problems with data security and access to databases or another organisation's resources.

A commonly referenced classification for challenges caused by GSD is [29][30]:

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

Communication breakdown (loss of communication richness). Human beings communicate best when they are communicating face-to-face. In GSD, face-to-face communication decreases due to distance, causing misunderstandings and lack of information over sites. For example, communication over distance can lead to

misinterpretation because people cannot communicate well due to language barriers.

Coordination breakdown. Software development is a complex process 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. This can cause problem solving to be delayed or the project to go down the wrong track until it becomes very expensive to fix. GSD 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 needs to be defined [31][32][33]. Coordination breakdown can also cause a number of specific problems; for example, Battin et al. [34] reported a number of software integration problems, which were due to a large number of independent teams. Wahyudin et al. [35] state that GSD demands more from project management. In addition to the project managers, the project members such as testers, technical leaders, and developers also need to be kept informed and notified of certain information and events that are relevant to their roles' objectives in timely manner which provides the conditions for in-time decision making.

Control breakdown (geographical dispersion). GSD means that management by walking around the development team is not feasible and, instead, telephones, email and other communication means (e.g., chat servers) must be used. These types of communication tools could be consider as less effective - not always providing 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 [34][36][37]. According to Holmstrom et al. [38], creating the overlap in time between different sites is challenging despite the flexible working hours and communication technologies that enable asynchronous communication. 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's success. Lack of informal communication, different processes and practices have a negative impact on teamness [31][32][34]. Furthermore, fear (e.g., of losing one's job to the other site) has direct negative impact on trust, team building co-operation and knowledge transfer, even where good relationships existed beforehand. According to Casey and Richardson [39] fear and lack of trust negatively impact 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 [40] 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 which leads to a more effective collaboration.

Culture clash (cultural differences). Each culture has different communication norms. In any cross-cultural communication the receiver is more likely to misinterpret messages or cues. Hence, miscommunication across cultures is usually present. Borchers [41] 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 [42], were found to be relevant from the software engineering viewpoint. Holmstrom et al. [38] discuss the challenge of creating a mutual understanding between people from different backgrounds. They concluded that often general understanding in terms of English was good, but more subtle issues, such as political or religious values, caused misunderstandings and conflicts during projects.

IV. MEASUREMENT BACKGROUND

In this section measurement background, the used terminology and traditional measurement methods, with GSD related challenges are introduced.

A. Traditional Metrics and Project Characteristics

Software measurements and metrics have been discussed since 1960's. The metrics have been classified many different ways. For example, they can be divided into basic and additional metrics [43] where basic metrics are size, effort, schedule and defects, and the additional metrics are typically metrics that are calculated or annexed from basic metrics (productivity = software size per used effort). The metrics can also be divided into objective or subjective metrics [43]. The objective metrics are easily quantified and measured, examples including size and effort, while the subjective metrics include less quantifiable data such as quality attitudes (excellent, good, fair, poor). An example of the subjective metrics is customer satisfaction. Furthermore, software metrics can be classified according to the measurement target, product, processes and resources [4]. Example metrics of product entities are size, complexity, reusability and maintainability. Example metrics of process entities are effort, time, number of requirements changes, number of specification/coding faults found and cost. Furthermore, examples of resource entities are age, price, size, maturity, standardization certification, memory size or reliability. These classifications, various viewpoints and the amount of examples merely prove how difficult the selection of metrics really can be during the project.

In addition to different ways of metrics classification, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparison should be done within the same kind of development projects, or the differences should be taken into account. Traditional project

characteristics are, for example; size and duration of a project, type of a project (development, maintenance, operational lifetime, etc.), project position (contractor, subcontractor, internal development etc.), type of software (hardware-related software development, application software, etc.) or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model etc.). Furthermore, different phases of development projects have to be taken consideration while analysing gathered measurement data.

B. Traditional Software Measurement and GSD

One of the most commonly used measurement methods at the end of 1990 and the beginning of 2000 was the Goal /Question /Metric (GQM) method. The GQM paradigm [3] represented a systematic approach for tailoring and integrating the objectives of an organisation into measurement goals and their step-wise refinement into measurable values. The GQM method was commonly known and was often used for searching and identifying organisations' strengths and weaknesses relating to the identified improvement goals. Furthermore, several assessment methods, for example CMMI [44] and SPICE (Software Process Improvement and Capability Determination, further known as a standard ISO/IEC 15504 Information technology — Process assessment), were generally used for identifying possible improvements areas and gaining knowledge of the software process of an organisation. In fact, the most of traditional measurements methods were based on expressions of the famous Shewhart cycle, called also the Deming cycle: PDCA (Plan–Do–Check–Act) [45]. The PDCA circle is an iterative four-step management method that is used in business for the control and continuous improvement of processes and products. The traditional methods used in software measurements were generally based on clearly defined and largely stabile processes that could be adjusted and improved. In those cases, the improvement actions were mainly done afterwards, for example, in the next project.

In GSD environment, where project stakeholders, work practices and development tools can vary by projects and partners, traditional measurement methods and actions are not adequate if they are used for process improvement purposes. There is little sense, if measurements only prove after the project what has happened during the project, because then it is too late to correct the situation. Furthermore, the lessons learned may not be suitable in the next projects. Overly large measurement programs with time consuming assessments are not worth paying the effort in dynamic GSD context. The traditional methods should be utilised for specific and well-aimed purposes. For example, the GQM method can be utilised while identifying new GSD specific metrics.

In GSD, development processes are dynamic and thus results of measurements and their interpretation vary. In this paper, GSD metrics used in the companies were focused on 'early warning' signals for the project and management. In a changing environment it's also an important aspect that the measurement data is easy to collect and that the metrics can be quickly calculated at regular intervals. Ease of use and speed are also central factors from metrics interpretation viewpoint. This also emphasises the importance of metrics visualisation. Interestingly, GSD literature has rarely focused on metrics and measurements or given experimental examples of successfully used metrics during GDS development.

C. Balancing Measurements

A Balanced Scorecard (BSC) is widely used for monitoring performance of an organisation towards strategic goals. The original BSC approach covers a small number of performance metrics from four perspectives, called as Kaplan & Norton perspectives: Financial, Customer, Internal Processes, Learning & Growth [46]. The BSC framework added strategic non-financial performance measures to traditional financial metrics to give managers and executives a more 'balanced' view of organizational performance. However, many early BSCs failed, because clear information and knowledge about the selection of measures and targets were not available. For example, organisations had attempted to use Kaplan & Norton perspectives without thinking about whether they were suitable in their situation. After that many improvements and enhancements have been completed on BSC approach. Since 2000, it has been described as a "Third Generation" of Balanced Scorecard designs. The BSC has evolved to be a strategic management tool that involves a wide range of managers in the strategic management process, provides boundaries of control, but is not prescriptive or constrictive and more importantly, removes the separation between formulation and implementation of strategy [47]. The BSC suggests that organisation should be viewed from four perspectives (Learning & Growth perspective, Business process perspective, Customer perspective, and Financial perspective) and metrics should be developed, data collected and analysed in relation to these perspectives.

Even if BSC are generally intended to deal with strategic issues, in this paper, the balancing of various perspectives of BSC has been emphasised. In fact, it has been proved that Practical Software Measurement and the Balanced Scorecard are both compatible and complementary [48]. In GSD context, decisions or actions taken based on the analysis of metrics and measurements collected from different development parties or stakeholders need to take specific the GSD factors into account as well.

D. Measurement Challenges in GSD

Even in the daily software development work, the measurements are still seen as unfamiliar or an extra burden for projects. For example, project managers feel it is time consuming to collect metrics for the organization (business-goal-related metrics), yet they need to have metrics that are relevant to the project. Furthermore, in many cases, not enough time is budgeted for measurements, and this is why it is very difficult to obtain approval from stakeholders for this kind of work [5].

Globally distributed development generates new challenges and difficulties for the measurements. For

example, the gathering of the measurements data can be problematic because of different development tools which have different versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. In addition, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that their comparison is impossible. Thus, the interpretation of measurements data is more complicated in GSD than one-site projects. This is why it is recommended to select a moderate amount of metrics. In this paper, we will present a set of metrics as well as examples of their visualisation possibilities to support decision making in GSD. Also industrial experiences about the metrics will be discussed.

The common metrics (effort, size, schedule, etc.) are also applicable for GSD projects. However, special attention may be needed in training the metrics collection, to ensure a common understanding of them (e.g., used classifications). In addition, as measurements also tend to guide people's behaviour, it is important to ensure that all are aware of the purpose of the metrics (i.e., not to measure individual performance), specifically in projects distributed over different cultures. In GSD content the automation of measurements is highly recommended to avoid misunderstanding - even if it is not easy to implement. The focus is to generate real-time information shown in a format that is easy and quickly interpreted. This means that great attention should be paid to metrics visualisation.

V. GENERIC MEASUREMENTS AND METRICS IN GSD

In this section, the metric set used in the companies is introduced. In addition, several visualised examples of proposed metrics are given and discussed. The metric set and their visualisation examples have been produced during the ITEA PRISMA (2008-2011) project [49]. The main goal of the PRISMA project was to boost productivity of collaborative systems development. One of the project's results was the Prisma Workbench (PSW), a tool integration framework [50]. PSW provides several real-time views into data that has been collected from various data sources even from separate stakeholders' databases. The PSW enabled the visualisation of metrics in GSD and collection of the experiences of their use. The work was done in close co-operation with industrial partners and experimental views were generated based on their needs or challenges. The original metrics were the same that the industrial partners had successfully used in their globally distributed projects, published in [1]. During the PRISMA project, the development of the PSW tool enabled further development of the proposed metrics set and their visualisation in co-operation with the industrial partners. The industrial partners had identified metrics, and defined their collection and visualisation. They had also tried the metrics in few projects to collect experiences. These experiences were then shared among the industrial partners of the project. The researchers analysed the measurements and experiences to find commonalities from these measurement practices. Results of this analysis was discussed in workshops with the companies, and updated based on the comments. This paper presents the results of this work. In following sub-sections, the developed example views are shown and discussed. Industrial experiences, opinions and ideas for improvement are also presented. The industrial experiences were gathered during the industrial cases by interviewing companies' personnel who had developed the metrics and measurement programs.

A. Rational Unified Process (RUP) Approach

Each phase in the lifecycle of a development project affects the interpretation of the metrics. Thus, in this paper, proposed metrics and visualisation examples are introduced by using commonly known approach of software development called Rational Unified Process (RUP). Also the processes used in the companies were similar to the RUP phasing, so it was chosen as a presentation framework for this paper. RUP is a process that provides a disciplined approach to assigning tasks and responsibilities within a development organisation. Its goal is to ensure the production of high quality software that meets the needs of its end-users within a predictable schedule and budget [16][51].

The software lifecycle is divided into cycles, each cycle working on a new generation of the product. RUP divides one development cycle in four consecutive phases [51]: (1) inception phase, (2) elaboration phase, (3) construction phase and (4) transition phase. There can be one or more iterations within each phase during the software generation. The phases and iterations of RUP approach are illustrated in following Figure 1.

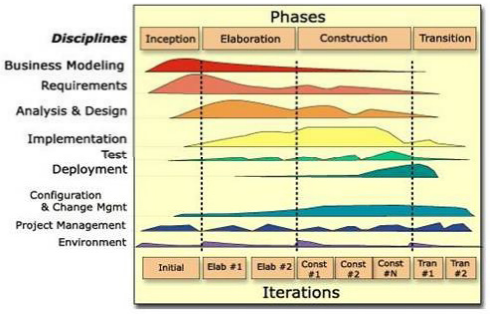

Figure 1. Phases and Iterations of RUP Approach [51]

From a technical perspective, the software development is seen as a succession of iterations, through which the software under development evolves incrementally [16]. From measurement perspective this means that some metrics can be focused on during one or two phases of the development cycle, and some can be continuous metrics that can be measured in all phases, and can be analysed in each iteration.

In this paper, the metrics are introduced according to the RUP phases. Each metric is presented in the phase where the

metric can be utilised in the first time or where the metric is seen to be the most relevant to measure, even if some metrics are relevant in several phases. In fact, many of the introduced metrics can be used also in the following product development phases. For each metric - a name, a notation and a detailed definition is introduced. The main goal is to offer a useful, yet reasonable amount of metrics, for supporting the on-time monitoring of the GSD projects. The indicators are supposed to be leading indicators rather than lagging indicators. For example, planned/actual schedule measurements should be implemented as milestone trend analysis which measures the slip in the first milestone and predicts the consequences for the other milestones and project end.

B. *Metrics and their Visualisation for Inception Phase*

During the inception phase, the project scope has to be defined and the business case has to be established. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones. Inception is the smallest phase in the project, and ideally it should be quite short. Example outcomes of the inception phase are a general vision document of the project's core requirements, main constraints, an initial use case model (10% -20% complete), and a project plan, showing phases and iterations [52]. Proposed metrics to be taken into consideration in this phase are introduced in Table I.

TABLE I. METRICS FOR THE INCEPTION PHASE

Metric	Notation	Definition
Planned Schedule	$D_{PLANNED}$	The planned Date of delivery (usually the completion of an iteration, a release or a phase)
Planned Personnel	# $FT_{PLANNED}$	The planned number of Full Time persons in the project at any given time
Planned Effort	$E_{PLANNED}$	The planned Effort for project tasks (/requirements) at any given time
Proposed Requirements	# Reqs	The number of proposed requirements.

The metrics Planned Schedule and Planned Personnel /Effort are mostly needed for comparison with actual schedule, personnel and effort, in order to identify lack of available resources as well as delays in schedule quickly. The amount of Proposed Requirements tells about the progress of the product definition.

Figure 2 shows how some of the proposed metrics can be utilised during product development for visualising the progress of project. The metric of progress status combines effort and schedule metrics in a visualised way. The first and top line (blue) in the Figure 2 is a cumulative planned effort over time calculated from project tasks. The next line, the red line describes the cumulative updated planned effort and accordingly, the green line describes the cumulative actual used effort over time summarised from project tasks. The bottom and last line in lilac shows the earned value that indicates the cumulative effort of completed tasks (/workproducts).

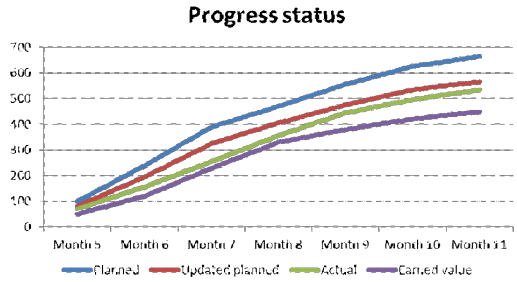

Figure 2. Visualised Metric: Progress Status

The graph visualises the project progress and easily gives several kinds of information as well as proactive insights, such as, is the project resourcing in place, and is the project completing work as planned. In the shown graph, it is a good signal that cumulative planned effort (blue line) is continuously above the cumulative updated planned effort (red line); it means the project is running on schedule. Another good signal is if actual used effort (green line) and earned value (lilac line) is relatively close to each others; it means that the results (~completed tasks) have been achieved with the used effort. The status in the Month 11 indicates that there are still several open tasks that are not completed even if actual used effort (green line) seems to draw closer to the cumulative updated planned effort (red line); this indicates a potential threat. Depending on project's phase (for example, in the middle phase or at the ending phase) corrective actions would be needed. The actions are not needed if the project is at the ending phase because the cumulative planned effort (blue line) is still clearly the upmost line.

Industrial comments

In the Philips company example, the Progress status metric has proven to give a timely insight in the actual consumption of effort compared to planned effort in large first-of-a-kind Consumer Electronics projects. The representation over time enables the ability to analyse trends, and take actions pro-actively. Moreover, the use of earned value gives insight in the effectiveness of the effort spent answering the question: "Does the effort spent contribute to realizing the agreed results?"

In the Symbio company example, indicators of earned value and tracking of unplanned work were seen as especially important from a management perspective. Unplanned work may yield a strong indication of a variety of causes early in the project, such as technical infeasibility or a lack of shared vision between project stakeholders. Accordingly, they identified that from a budget perspective, justifying workshops early in the project to shape a shared vision and collaborate on scoping project goals is often difficult to qualify for many stakeholders. It is a typical case that only when problems manifest, or a sharp trend in unplanned work is experienced will stakeholders react. Usually, remedying the problem requires unplanned trips to

put people together into the same room to hammer out solutions that essentially consume budget.

C. Metrics and their Visualisation for Elaboration Phase

During the elaboration phase a majority of the system requirements are expected to be captured. The purpose of the phase is to analyse the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. The final elaboration phase deliverable is also a plan (including cost and schedule estimates) for the construction phase. Example outcomes of the elaboration phase are; a use case model (at least 80% complete), a software architecture description, supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case, a revised risk list and a revised business case, and a development plan for the overall project. Proposed metrics to be taken into consideration in this phase are introduced in Table II.

TABLE II. METRICS FOR THE ELABORATION PHASE

Metric	Notation	Definition
Schedule: Planned /Actual Schedule	 **D**$_{PLANNED}$ **D**$_{ACTUAL}$	The planned/actual Date of delivery (usually the completion of an iteration, a release or a phase)
Staff: Planned /Actual Personnel Planned /Actual Effort	 **#FT**$_{PLANNED}$ **#FT**$_{ACTUAL}$ E$_{PLANNED}$ E$_{ACTUAL}$	The planned/actual number of Full Time persons in the project at any given time. The planned/actual Effort for project tasks (/requirements) at any given time.
Requirements -Drafted -Proposed -Approved -Not implemented	 **#Reqs** $_{DRAFTED}$ **#Reqs** $_{PROPOSED}$ **#Reqs** $_{APPROVED.}$ **#Reqs** $_{NOT\_IMPL}$	The number (#) of - drafted requirements - proposed requirements - reqs approved by customer - not implemented reqs
Tests -Planned	 **#Tests** $_{PLANNED}$	The number (#) of - planned tests
Documents: -Planned -Proposed -Accepted	 **#Docs** $_{PLANNED}$ **#Docs** $_{PROPOSED}$ **#Docs** $_{ACCEPTED}$	The number (#) of planned /proposed /accepted documents to be reviewed during the project.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. The Staffing metric may explain deviations in the expected progress vs. the actual progress, both from a technical as well as from a schedule viewpoint. Note that those metrics that are more relevant to measure by iterations (effort and size) are introduced later (in Section E).

Figure 3 shows how some of the proposed metrics can be visualised in order to describe the project's status. The metric of requirements status combines the amount of planned effort with status of requirements' implementation over a time in the same graph. The bars summarise the amount of planned effort for the month. Each bar is composed from four different data relating to identified requirements as follows. The first block (green) describes a sum of planned efforts for all implemented requirements. The second block (grey) describes a sum of planned efforts for approved but not implemented requirements. The third block (blue) decribes a sum of planned efforts for proposed requirements and the last block (orange) shows a sum of planned efforts for drafted requirements.

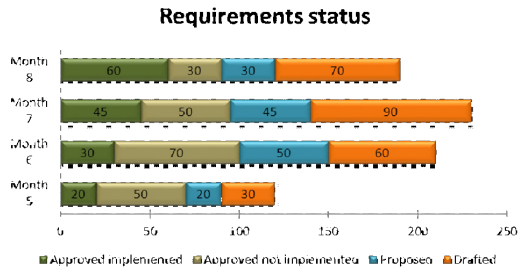

Figure 3. Visualised Metrics: Requirements Status

It is important to note, that the planned effort is used constantly, even for implemented requirements. This is due to keeping the baseline in order to enable comparing project situation over time, i.e., to be able to see the project trend with respect to planned work. The planned effort may be updated for the requirements during the project, if a new baseline is created. This information is then used together with the actuals, to see how well the planning has succeeded to help learning to estimate better.

The visualised metric "Requirements status" indicates several status information but also trend lines relating to requirements implementation, and is focused on showing the uncertainly of the project, for example how much more work maybe dedicated to be implemented in the project. In the example graph, a good signal is that the sum of planned efforts for implemented requirements seems to increase over time while the sum of planned efforts for approved, but not implemented requirements, seems to reduce. However, the sums of planned efforts for proposed and drafted requirements are still quite large in the Month 8, especially, while comparing them to the sums of planned efforts for approved requirements. This indicates that the project is in the beginning phase rather than in the ending phase. However, the interpretation needs other metrics information, such as "Progress status" or "Testing status" to make any decisions.

Industrial comments

In the Philips company example, the current projects lack insight into the satisfaction of requirements. This lack of insight concerns both the actual status of implementation of the requirements, as well as the expectation: "Up to what level the project will be able to satisfy its requirements, and if not, what are measures to accomplish that?" The (leading) indicator as proposed in this document seems to be a good answer to this problem. The metric has been introduced in a few (one-roof) projects yet and initial results seem promising. However, no data with experiences on a metric like this have been collected yet.

According to Symbio's practice, when looking to exit an elaboration phase, product owners should pay special

attention to the coverage of requirements affecting architecture to ensure the construction phases run more to plan as the team sizes may scale and involve more sites. Whilst iterative development can be seen as promoting elaboration of requirements later in the lifecycle, core functions that separate the project output from competition should be conceptualized and approved for implementation. Project managers may consider implementation of these differentiating use cases to be made geographically or temporally close to the project owner. Non-approved requirements should be managed accordingly and not planned for implementation off-site until they are suitably elaborated and accepted into the development roadmap. Misunderstanding of the requirements needs to be minimalized if the team size and development sites scale during construction phases otherwise projected cost savings from multi-site development can be quickly eliminated.

D. Metrics and their Visualisation for Construction Phase

Construction is the largest phase in the project. During the phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. Example outcomes of the phase consist of a software product integrated on the adequate platforms, user manuals, and a description of the current release. Proposed metrics to be taken into consideration in this phase are introduced in Table III.

TABLE III. METRICS FOR THE CONSTRUCTION PHASE

Metric	Notation	Definition
Planned /Actual Schedule Planned /Actual Personnel	$\mathbf{D}_{\text{PLANNED}}$ $\mathbf{D}_{\text{ACTUAL}}$ $\mathbf{\#FT}_{\text{PLANNED}}$ $\mathbf{\#FT}_{\text{ACTUAL}}$	Defined in the elaboration phase.
Requirements: -Proposed -Approved -Not implemented -Started -Completed	 $\mathbf{\#Reqs}_{\text{PROPOSED}}$ $\mathbf{\#Reqs}_{\text{APPROVED}}$ $\mathbf{\#Reqs}_{\text{NOT\_IMPL}}$ $\mathbf{\#Reqs}_{\text{STARTED}}$ $\mathbf{\#Reqs}_{\text{COMPLETED}}$	The number (#) of - proposed requirements - reqs approved by customer - not implemented reqs - reqs started to implement - reqs completed
Change Requests: -New CR -Accepted -Implemented	 $\mathbf{\#CRs}_{\text{NEW}}$ $\mathbf{\#CRs}_{\text{ACCEPTED}}$ $\mathbf{\#CRs}_{\text{IMPL}}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented
Tests: -Planned -Passed -Failed -Not tested	 $\mathbf{\#Tests}_{\text{PLANNED}}$ $\mathbf{\#Tests}_{\text{PASSED}}$ $\mathbf{\#Tests}_{\text{FAILED}}$ $\mathbf{\#Tests}_{\text{NOT TESTED}}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
Defects -by Priority: e.g., Showstopper, Medium, Low	$\mathbf{\#Dfs}_{\text{PRIORITY}}$	The number (#) of - defects by Priority during the time period
Documents: -Planned -Proposed -Accepted	 $\mathbf{\#Docs}_{\text{PLANNED}}$ $\mathbf{\#Docs}_{\text{PROPOSED}}$ $\mathbf{\#Docs}_{\text{ACCEPTED}}$	Defined in the elaboration phase.

Note that those metrics that are continuously measured are introduced later (in Section E).The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Defect metrics describe both the progress of testing as well as the maturity of the product.

In the construction phase, all components and features are developed and integrated into the product. In addition, they are also thoroughly tested, so there are many simultaneous actions that can be implemented by multiple partners or/and in different locations in GSD. This is why the metrics interpretation needs to be done very carefully by utilising indicators from different data sources and from different partners. In this subsection two metrics: "Budget status" and "Testing status" are introduced with discussion about indicators and proactive signals that they provide.

The visualisation of Budget status combines cost, requirements and defects metrics in the same graph shown in Figure 4.

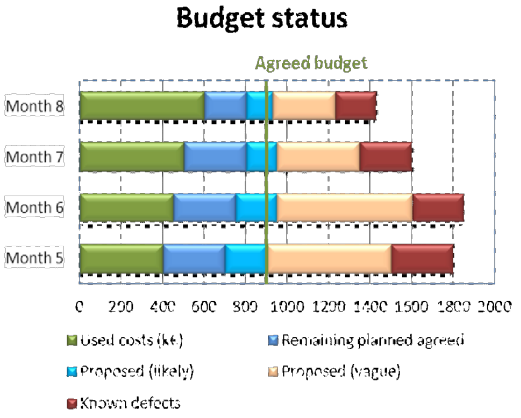

Figure 4. Visualised Metrics: Budget Status

The Budget status graph shows actual costs of the project in portion with the agreed budget over a time period. The metric also gives several indicators of estimated prospective costs in each month. The bars summarise amount of costs for the month, and each bar is composed from five different cost-related data. The first block (green) describes actual cumulative costs of the project. The agreed budget for the project is shown clearly as a green line in the middle of the graph. The second block (blue) describes remaining planned cost based on effort estimated for requirements that have been accepted for implementation but not yet implemented. The third block (light blue), in the middle of the bar, indicates proposed cost that can be seen very likely costs for the project. These costs are based on effort estimated for the proposed requirements that are estimated likely to be implemented, for example, a customer will want them. The fourth block (orange) describes proposed but vague costs for

the project. These costs are based on effort estimated for the proposed requirements which the likeliness for implementation is not known. Instead, the fifth block (red) indicates very potential costs for the project, so-called "Known defects" costs. The costs are based on effort estimated to be needed to fix the known critical, major or average defects. In the example graph, the Budget status metric in Figure 4, the project's costs will overrun the agreed budget.

Industrial comments

At Philips, the current applied budget metrics generally give a clear understanding in the actual budget consumption, but are poor in predicting budget consumption for the remainder of the project. The metric suggested allows for trend analysis and by that extrapolation to the future, resulting in better prediction of the budget consumption for the remainder of the project. This will improve the projects' and the management's insight into the project and enable them to take required measures in a timely fashion, as appropriate. The metric has not yet been applied in our projects.

In Symbio, managers will often track cost against budget throughout construction for project sponsors, but earned value becomes increasingly more important in the latter stages of the lifecycle. Earned value can be tracked with relative ease if defined requirements are quantified for business importance. Product backlogs imply the importance by a requirement's position in the backlog; however some backlogs may include other items than requirements such as operational tasks for deployment and so on. To compensate all project requirements (both functional and non-functional) can be attributed with a business value, its value based in comparison to the cumulative value of all project requirements. When a requirement is delivered its value is added to cumulative total to provide an earned value delivered by the project. This approach is ideal if the backlog of the product development stabilizes throughout construction. However significant changes in the business value of requirements will weaken the importance of tracking this metric over time. Also this metric requires the project team and stakeholders to agree upon a "definition of done" which can be very difficult, and even more so if the accepting and implementing parties are different entities or located in different sites.

The metric of Testing status combines effort, requirements and test metrics in a same graph. The Testing status metric visualises the progress of testing phase by collecting data from various phases. The bars in the graph summarise efforts relating to tests in each month. Each bar is composed from four different sums of efforts. The first block (green) describes a sum of efforts for tested requirements. The second block (blue) describes a sum of efforts for requirements for which test case is available, and accordingly, the third block (purple) describes a sum of efforts for requirements for which test cases are not available. The last, the fourth block (red) is a very proactive indicator describing a sum of effort estimated for uncertain requirements. Figure 5 shows the visualisation of Testing status metric.

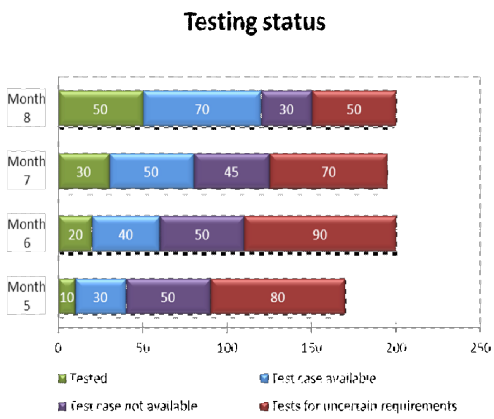

Figure 5. Visualised Metrics: Testing Status

Even if "Testing status" shows easily how 'mature' the testing phase is the metric requires other metrics – such as the before introduced metrics: Budget status, Progress status and Requirements status – make conclusions based on the data.

Industrial comments

According to Philips, one of the most important indicators of a development project is insight in what will be the status of the product at the delivery time - what will the product actually contain and what is the quality of those contents? This metric is an effective means to get early insight in the status of the product by the end of the project. Moreover, the test status trend analysis helps to initiate timely measures to work towards an agreed project result. The metric has been applied in a single project at Philips and results were promising - it really improved the insight of project, management and customer in the status of the product-under-construction and better understanding of what could be expected by the end of the project.

According to Symbio, earned value is especially invaluable in the close down phases of a project. Projects may deteriorate into loss making, unplanned iteration as stakeholders become overly conscious on metrics of requirements coverage. This situation is can be further exacerbated if the value of requirements is not continually reviewed and communicated to all stakeholders throughout the project.

E. Metrics for Transition Phase

The final project phase of the RUP approach is transition. The purpose of the phase is to transfer a software product to a user community. Feedback received from initial release(s) may result in further refinements to be incorporated over the course of several transition phase iterations. The phase also includes system conversions, installation, technical support, user training and maintenance. From measurements viewpoint the metrics identified in the phases relating to schedule, effort, tests, defects, change requests and costs are still relevant in the transition phase. In addition, customer

satisfaction is generally gathered in the transition phase, and post-mortem analysis carried out.

F. Metrics for Iterations

Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release. Each release is accompanied by supporting artifacts: release description, user's documentation, plans, etc. Although most iterations will include work in most of the process disciplines (requirements, design, implementation, testing) the relative effort and emphasis will change over the course of the project. Proposed metrics for each iteration to be taken into consideration, are introduced in Table IV.

TABLE IV. METRICS FOR ITERATIONS

Metric	Notation	Definition
Effort: -Planned Effort -Actual Effort	 $\mathbf{E}_{PLANNED}$ $\mathbf{E}_{ACTUAL}$	The planned/actual effort required of any given iteration of the project.
Size: -Planned size -Actual size	 $\mathbf{SIZE}_{\text{PLANNED}}$ $\mathbf{SIZE}_{\text{ACTUAL}}$	The planned /actual size of each iteration can be measured as SLOC (Source Lines of Code), Points or any other commonly accepted way.
Cost: -Budgeted -Expenditure	 $\mathbf{COST}_{\text{BUDGET}}$ $\mathbf{COST}_{\text{ACTUAL}}$	The budgeted cost /actual expenditure for any given iteration.
Velosity: -planned /actual story points	 $\mathbf{\#PTS}_{\text{PLAN}}$ $\mathbf{\#PTS}_{\text{ACT}}$	How many story points are planned to be /actually implemented of any given iteration of the project.
Productivity:	$\mathbf{E}_{ACTUAL}$ / $\mathbf{\#PTS}_{\text{ACT}}$	Use effort per acutally implemented story points for each sprint /iteration

All of these metrics provide indications of the project progress and reasons for deviations should be analysed. These metrics should be analysed together with other metrics results (presented in Tables I-III) in order to gain a comprehensive picture of the status.

VI. SPECIFIC MEASUREMENTS AND METRICS IN GSD

Section V discussed metrics, which are not specific for GSD, but they provide valuable information to follow a GSD project progress. So, in GSD, metrics can be similar or same as in single-site development. However, in order to prevent potential problems during distributed projects some specific GSD metrics could be added to be used together with the metrics presented in Section V. These metrics should be focused on the specific challenges in GSD that were presented in general level in Section III and they would help to quickly detect the GSD related source of problems that are identified in the metrics presented in Section V.

Measuring the generic GSD challenges (Section III) is difficult, and in fact, measuring the challenges does not provide clear value from project monitoring viewpoint. It is more beneficial to follow and detect the symptoms that indicate problems in the GSD practice. Example problems [14] caused by lack of communication, coordination breakdown, and different backgrounds 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.

Example problems [14] caused by lack of teamness and lack of trust include hiding problems and 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.

The following problems are among the most common ones in companies GSD practice (based on 54 industrial cases during several research projects):

1. unclear responsibilities and escalation channels,
2. unavailability of information timely for all who need it,
3. unclear information and misunderstandings (for example of requirements and task assignments),
4. problem hiding,
5. non-communicated and unexpected changes,
6. lack of visibility and transparency of all sites work and progress,
7. faulty and/or delayed (internal) deliveries, and
8. sub optimal use of resources.

Next we discuss potential measurements to indicate as early as possible if these problems are present. These proposals have not been applied in practice, yet. Instead, their implementations and possible selections were discussed with industrial partners.

Relating to problems 1-3, a measurement could be a short questionnaire asking the project members if they know their responsibilities and when and to whom to escalate problems, and is the required information available and clear. In addition, from GSD viewpoint, a potential measurement could be related to time spent idling (a team member is waiting because of wrong, incorrect or missing information or input from other members) or percentage of unplanned work (a team member is working with unplanned or duplicated tasks).

For problems 4 – 6, a measure is amount and type of communication over sites. For example, communications activeness could be monitored via metrics like amount of status reports, meeting memos, chats, calls between

locations, etc. Communication activeness is especially important between distributed teams where their development tasks are highly coupled or dependent on deliveries and results of each other. For example, silence or communication only via documents (official reports) can be an indicator of problem, whereas active informal communication over sites indicates active discussion of work at hand. In the worst case in GSD, lack of face-to-face communication can lead to "reportmania" where communication is handled only through large amount of documents. Long textual descriptions can be easily omitted or alternatively misunderstood because of high amount of effort and time required for adopting the content.

For problem 7, metrics related to defects and schedule are relevant and for problem 8, a potential measurement is time spent idling and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when a team is not performing. Also, communication related metrics are valuable for these problems.

The metrics relating to team trust, project commitment and team identifications describes team dynamics that can provide lot of explaining information for the problems in GSD project. Some indicators, such as how many people have left from the project, refer the individual satisfaction as well as project commitment. Because software development is fundamentally team oriented action [53], metrics relating to team dynamics and teamwork quality is highly recommended to monitor in GSD. Potential metrics are related to communication, tasks coordination, balance of member contributions, mutual support, effort and cohesion as introduced by Hoegl and Gemuenden [54]. Examples of questions are as follows:

- Communication: Is there sufficient frequent, informal, direct, and open communication?
- Coordination: Are individual efforts well-structured and synchronised within the team?
- Balance of member contributions: Are all team members able to bring in their expertise to its full potential?
- Mutual support: Do team members help and support each other in carrying out their tasks?
- Effort: Do team members exert all efforts to team tasks?
- Cohesion: Are team members motivated to maintain the team? Is there team spirit?

These questions can be used to measure team dynamics and team work quality during a GSD project.

VII. DISCUSSION

A. GSD Metrics

As discussed, little focus has been paid on GSD metrics in the literature. In fact, the research has been focused on clarifying differences between collocated and distributed projects and also, identifying variables that differ the most. Although this kind of approach is important for gaining knowledge about the issues that need to be monitored in GSD, a specific focus on the metrics and their collection and analysis is also needed. For example, project performance is even more complicated and multi-level concept to measure in GSD than in single-site. It concerns team members' individual performance, teamwork performance and tasks performance as well as management performance. Bourgault et al. [19] pointed out that distributed projects' performance metrics and measurement needs more attention so that well designed management information systems could be developed in order to create effective monitoring systems for distributed projects. This kind of development was seen as necessary to provide decision makers with dynamic, user-friendly information system that would support management activities, not only for project managers, but also for top managers. However, the issue of performance metrics in the context of distributed projects needs to be investigated in more detail. Furthermore, a dispersion of work has significant effects on productivity and, indirectly, on the quality of the software. However, it is currently difficult to specify metrics, measurement processes and activities that best suit different companies and specific GSD circumstances. We have presented a first step towards taking into account the specific aspects of GSD in measurement programs, but more work is needed. For example, specific GSD metrics are currently collected and processed manually, thus requiring extra and error prone effort. In the world of the hectic and dynamic GSD practice, the metrics collection and visualisation should also be automated to be valuable in large-scale use. The automation is an important issue for further research.

B. Industrial Viewpoint

The metrics presented in Section V were common for both of the companies. Although the metrics were chosen independently by both companies, the reasoning behind choosing these metrics was similar. An important reason was to come from a re-active into a pro-active mode, for example to introduce 'early warning' signals for the project and management. Specifically these metrics have been chosen as they indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort, distributed over sites and companies. Accordingly, the metrics set can be seen as a 'balanced score card', on which management can take the right measures, balancing insights from time, effort (e.g., staffing), cost, functionality (requirements) and quality (tests) perspective.

An important aspect was also that the metrics are easy to capture and that they can be captured from the used tools "for free", or can be quickly calculated at regular intervals. Costs and budgets are good examples of metrics that can be easily captured from the tools. This is also important from GSD viewpoint, as automated capturing reduces the chance of variations caused by differences in recording the metrics data in different sites. Neither of the companies use metrics based on lines-of-code as they did not find it to be a reliable indicator of progress, size or quality of design.

It can be seen that the metrics are quite similar as in single-site development. However, the metrics may be analysed separately for each site, and comparisons between sites can thus be made in order to identify potential problems early. On the other hand, it is important to recognise that some metrics correlate with each other, for example, metrics relating to tests correlate with metrics about requirements, and that needs to take consideration while analysing. In general, the interpretation of project's comprehensive status needs various metrics information – like Requirements status, Progress status, Testing status and Budget status – for making conclusions based on the data. In addition, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires 'super-balancing' - how to come to the right corrective action if for instance, on the one side, the % of not accepted requirements is high, and on the other side, the # of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, relating to subjective metrics, such as effort estimation, differences between backgrounds of the people (cultural or work experience) in different sites may affect the result.

The companies also use the measurement results to gain insight into why a measure varies between similar single-site and multi-site projects in order to try to reduce potential variances. This also partially explains the use of the same metrics as single-site development. This was experienced by the representative of Symbio: "*These points presented should be by now well known. From an economic perspective these points must be considered when evaluating and comparing costs of different project models of delivery."*, and *"Benchmarking and tracking of historical data across the entire project portfolio is still only an initial step to shape more informed cost estimations when composing project teams with distributed elements. Continuous effort is required not only in definition and capture of metrics but also in the effects on working practices in general.*"

Furthermore, the challenges in communication and dynamics of distributed teams mean that working practices need to be addressed continuously as impressed by Symbio representative: "*Often a practical solution to working procedures can result in compensation for potential lost productivity. For example a testing team in China lags their working week by one day (Tuesday to Saturday) in order to test the results from an implementation team in Finland (working Monday to Friday). In this example the Finland team agrees to ensure continuous integration in order to not block the testing team. If these two practices have a positive effect on productivity when compared against similar project models, future cost estimations should then be benchmarked on the new working practices.*" However, in addition to metrics results, paying close attention and acting on feedback is as important, if not more important than drawing strong conclusions from metrics alone.

Currently, both companies are in process of revamping their metric usage, but feel confident that the metrics introduced in this paper are the right ones. This was pointed out by Philips by the following: "*Applying the metrics suggested in this document to the parties involved in the GSD project already gives better insight in the relative performances of the groups, and enables to take measures over time (e.g., systematically improve a party's performance, or replace it). We have applied detailed effort consumption metrics to our single-roof and multi-side development projects. Those metrics learned that staff of multi-side projects spend significantly more time on things they call 'communication' or 'overhead' (up to 50%!). Our understanding of the matter is that no new metric needs to be 'invented' for that: standard effort distribution metrics would do. The main challenge is to have it introduced in a systematic way, with the same understanding and interpretation of the metrics by the parties involved. Especially the first element is often a challenge: third parties are often reluctant to provide this level of transparency of their performance.*"

Both companies are careful in introducing new metrics, as it is well known that too many metrics lead to overkill and rejection by the organization, and do not provide the right insights and indication for control measures. Easy implementation and by that, easy acceptance is the most crucial thing to get these metrics as established practice within the company. However, the few specific GSD metrics presented in Section VI are intended to be used together as the proposed metrics set. These additional metrics should be focused on measuring the project performance, especially task and team performance in GSD.

VIII. CONCLUSION

The management of the more and more common distributed product development project has proven to be more challenging and complicated than traditional one-site development. Metrics are seen as important activities for successful product development as they provide the means to effectively monitor the project progress. However, defining useful, yet reasonable amount of metrics is challenging, and there is little guidance available for a company to define metrics for its distributed projects.

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools and their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. Furthermore, interpretation and decision-making based on the measurement results require that the distributed development implications are taken carefully into consideration.

This paper focused on describing a set of metrics that is successfully used in industrial practice in GSD and given examples of their visualisation with industrial experiences of their use. These metrics, are aimed especially to provide the means to proactively react to potential issues in the project, and are meant to be used as a whole, not interpreted as single information of project status. The basic GSD circumstances with challenges are discussed from viewpoints of metrics

and measurements in order to create awareness and knowledge of potential GSD specific metrics.

The metrics presented in the paper were common for both of the companies. Based on experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed at regular intervals. Also, one of the most important reasons was that these metrics were aimed especially to provide the means to proactively react to potential issues in the project. The balancing insights from time, effort, cost, functionality and quality was also seen as very important aspect.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024 [49]. The authors would like to thank the support of ITEA [55] and Tekes (the Finnish Funding Agency for Technology and Innovation) [56].

REFERENCES

[1] M. Tihinen, P. Parviainen, R. Kommeren and J. Rotherham, "Metrics in distributed product development," In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11), Barcelona, Spain, 2011, pp. 275-280.

[2] R. Van Solingen and E. Berghout, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill, 1999.

[3] V. R. Basili, Software modeling and measurement: The Goal /Question/Metric paradigm. Computer Science Technical Report CS-TR-2956, UNIMACS-TR-92-96, University of Maryland at College Park, Sep. 1992, pp. 1-24.

[4] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co. Boston, MA, USA, 1998.

[5] M. Umarji and F. Shull, "Measuring developers: Aligning perspectives and other best practices," IEEE Software, vol. 26, (6), 2009, pp. 92-94.

[6] J. Hyysalo, P. Parviainen and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," In Proceedings of 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.

[7] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," In Proceedings of Future of Software Engineering FOSE '07, IEEE Computer Society, 2007, pp. 188-198.

[8] J. D. Herbsleb, A. Mockus, T. A. Finholt and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, 2000, pp. 319-328.

[9] M. Jiménez, M. Piattini and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," Advances in Software Engineering, vol. Jan-2009, (No. 3), 2009, pp. 1-16.

[10] S. Komi-Sirviö and M. Tihinen, "Lessons learned by participants of distributed software development," Knowledge and Process Management, vol. 12, (2), 2005, pp. 108-122.

[11] M. Tihinen, P. Parviainen, T. Suomalainen, K. Karhu and M. Mannevaara, "ABB experiences of boosting controlling and monitoring activities in collaborative production," In Proceedings of the 6th IEEE International Conference on Global Software Engineering (ICGSE'11) Helsinki, Finland, 2011, pp. 1-5.

[12] F. Q. B. da Silva, C. Costa, A. C. C. França and R. Prikladinicki, "Challenges and solutions in distributed software development project management: A systematic literature review," In Proceedings of International Conference on Global Software Engineering (ICGSE2010), IEEE, 2010, pp. 87-96.

[13] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," In Proceedings of the15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.

[14] P. Parviainen, "Global software engineering. challenges and solutions framework," Doctoral Dissertation, VTT Science 6, Finland, 2012, pp. 106 p. + app. 150 p.

[15] Prisma-wiki, SameRoomSpirit wiki homepage. URL: http://www.sameroomspirit.org/index.php/Main_Page (Accessed 19.12.2012).

[16] P. Kruchten, The Rational Unified Process: An Introduction. Addison-Wesley Professional, 2004.

[17] C. E. L. Peixoto, J. L. N. Audy and R. Prikladnicki, "Effort estimation in global software development projects: Preliminary results from a survey," In Proceedings of International Conference on Global Software Engineering, IEEE Computer Society, 2010, pp. 123-127.

[18] K. Korhonen and O. Salo, "Exploring quality metrics to support defect management process in a multi-site organization - A case study," In Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2008, pp. 213-218.

[19] M. Bourgault, E. Lefebvre, L. A. Lefebvre, R. Pellerin and E. Elia, "Discussion of metrics for distributed project management: Preliminary findings," In Proceedings of the 35th Annual Hawaii International Conference on System Sciences HICSS'02, IEEE, 2002, 10 p.

[20] S. Misra, "A metric for global software development environment," In Proceedings of the Indian National Science Academy 2009, pp. 145-158.

[21] R. M. Lotlikar, R. Polavarapu, S. Sharma and B. Srivastava, "Towards effective project management across multiple projects with distributed performing centers," In Proceedings of IEEE International Conference on Services Computing (CSC'08), IEEE, 2008, pp. 33-40.

[22] M. T. Lane and P. J. Ågerfalk, "Experiences in global software development - A framework-based analysis of distributed product development projects," In Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009). 2009, pp. 244-248.

[23] A. Piri and T. Niinimaki, "Does distribution make any difference? quantitative comparison of collocated and globally distributed projects," In Proceedings of the Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW'11), 2011, pp. 24-30.

[24] B. Sengupta, S. Chandra and V. Sinha, "A research agenda for distributed software development," In Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 731-740.

[25] N. Ramasubbu and R. K. Balan, "Globally distributed software development project performance: An empirical analysis," In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference aNd the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), ACM, 2007, pp. 125-134.

[26] D. B. Simmons, "Measuring and tracking distributed software development projects," In Proceedings the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003). IEEE, 2003, pp. 63-69.

[27] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," In Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), IEEE, 2006, pp. 33-38.

[28] I. A. da Silva, M. Alvim, R. Ripley, A. Sarma, C. M. L. Werner and A. van der Hoek, "Designing software cockpits for coordinating distributed software development," In the First Workshop on

Measurement-Based Cockpits for Distributed Software and Systems Engineering Projects, 2007, pp. 14-19.

[29] E. Carmel, Global Software Teams: Collaborating Across Borders and Time Zones. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[30] E. Carmel and P. Tija, Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce. Cambridge University Press, the United Kingdom, 2005.

[31] D. E. Damian and D. Zowghi, "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations," In Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), 2003, 10 p.

[32] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," IEEE Transactions on Software Engineering, vol. 29, (6), 2003, pp. 481-494.

[33] M. Paasivaara and C. Lassenius, "Collaboration practices in global inter-organizational software development projects," Software Process: Improvement and Practice, vol. 8, (4), 2003, pp. 183-199.

[34] R. Battin, R. Crocker, J. Kreidler and K. Subramanian, "Leveraging resources in global software development," IEEE Software, vol. 18, (2), 2001, pp. 70-77.

[35] D. M. Wahyudin, S. Heindl, A. Biffl and B. R. Schatten, "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, 2007, pp. 20-25.

[36] J. D. Herbsleb and D. Moitra, "Global software development," IEEE Software, vol. 18, (2), 2001, pp. 16-20.

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

[38] H. Holmstrom, E. O. Conchuir, P. J. Ågerfalk and B. Fitzgerald, "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), IEEE, 2006, pp. 3-11.

[39] V. Casey and I. Richardson, "Virtual teams: Understanding the impact of fear," Software Process Improvement and Practice, vol. 13, (6), 2008, pp. 511-526.

[40] B. Al-Ani and D. Redmiles, "Trust in distributed teams: Support through continuous coordination," IEEE Software, vol. 26, (6), 2009, pp. 35-40.

[41] G. Borchers, "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, 2003, pp. 540-545.

[42] G. Hofstede, Culture's Consequences. Comparing Values, Behaviors, Institutions, and Organizations, Across Nations. Sage Publications. London, 2nd edition, 2001.

[43] K. H. Möller and D. J. Paulish, Software Metrics: A Practitioner's Guide to Improved Product Development. Institute of Electrical & Electronics Enginee, London, 1993.

[44] CMMI, "CMMI for development," Tech. Rep. version 1.2., Technical Report CMU/SEI-2006-TR-008, 2006.

[45] W. A. Shewhart, Statistical Method from the Viewpoint of Quality Control. Graduate School of Agriculture, Washington, 1939. Referenced in W.E. Deming: Out of Crisis. Cambridge, Mass.: MIT Center for Advanced Engineering Study, 1986.

[46] R. S. Kaplan and D. P. Norton, "The balanced scorecard-measures that drive performance," Harward Business Review, (No. 92105), 1992, pp. 71-79.

[47] G. Lawrie and I. Cobbold, "Third-generation balanced scorecard: Evolution of an effective strategic control tool," International Journal of Productivity and Performance Management, vol. 53, (7), 2004, pp. 611-623.

[48] D. Card, "Integrating practical software measurement and the balanced scoreboard," In Proceedings of the 27th Annual International Computer Software and Applications Conference COMPSAC 2003, 3-6 Nov. 2003, pp. 362- 363.

[49] PRISMA, Productivity in Collaborative Systems Development, ITEA project (2008-2011) number 07024, Project info page, URL: http://www.itea2.org/project/index/view/?project=237 (Accessed 19.12.2012).

[50] J. Eskeli, J. Maurolagoitia and C. Polcaro, "PSW: A framework-based tool integration solution for global collaborative software development," In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11), Barcelona, Spain, 2011, pp. 124-129.

[51] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley Pub Co, Addison-Wesley Object Technology Series, 1999.

[52] P. Kruchten, "A rational development process," CrossTalk, vol. 9, (7), 1996, pp. 11-16.

[53] E. Demirors, G. Sarmasik and O. Demirors, "The role of teamwork in software development: Microsoft case study," In Proceedings of the 23rd EUROMICRO Conference, New Frontiers of Information Technology, 1997, pp. 129-133.

[54] M. Hoegl and H. G. Gemuenden, "Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence," Organization Science, vol. 12, (4), 2001, pp. 435-449.

[55] ITEA 2, Information Technology for European Advancement, ITEA 2 homepage, URL: http://www.itea2.org/ (Accessed 19.12.2012).

[56] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: http://www.tekes.fi/eng/ (Accessed 19.12.2012).

Series title and number

VTT Science 70

Title	**Measurement-based management of global software development projects**
Author(s)	Maarit Tihinen
Abstract	Collaborative, global software development has become the norm in software-intensive systems development. Because software products are developed in such dynamic environments, where requirements, priorities, participating sites, development processes and tools, and even partners are continuously changing, project control and management activities are increasingly important. In fact, up-to-date information of project status is now critical to completing project management effectively. However, management of a global software development project is more challenging than traditional development. Thus, project management and coordination mechanisms are important means for avoiding project failure in global software development. Measurements and metrics create useful ways for controlling and managing a project during product development. In addition, measurements bring several benefits for an organisation (e.g., better time-to-market estimation via improved project and product management, better control over product development costs owing to a more visible development process, higher sales, and improved customer satisfaction because of higher product quality). However, in practise, organisations usually cannot allocate enough time nor resources to plan and carry out the measurements properly. In fact, in daily project work, measurements and metrics can be experienced as unfamiliar. In global software development, partners and stakeholders change according to each new collaborative setting, and this brings new challenges for measurements, which means that one solution is not appropriate for all situations. Instead, while defining metrics and measurements in global software development, the larger picture must be considered. In this thesis, the main challenges related to measurements and metrics in global software development projects have been studied and analysed in detail. In addition, development tools and methodologies-related challenges are identified and analysed to draw insights to create a proof-of-concept solution to be implemented. This tool-integration solution was developed for supporting project management with automated and real-time indicators during global software development projects. The thesis pointed out that automatically produced real-time metrics, whose original measurement data are gathered from various databases, even from different stakeholders, are a robust and feasible method to produce valuable, reliable, and up-to-date information for decision making in global software development projects. The thesis provides a definition for dynamic measurements, where metrics are defined based on demands of each project's collaboration settings, metrics data is collected and analysed continuously from various tools and databases, and measurement data is analysed and visualised for easy to read format. The implemented proof of concept tool integration solution enabled the researcher to study the benefits and industrial experiences of dynamic measurements. The thesis concluded that dynamic measurements are highly beneficial for managing a global software development project. Based on the case studies, dynamic measurements were proved to be feasible via the use of automated and real-time metrics with consolidated information via visualised dashboards. The thesis concluded that the measurement-based management of global software development projects is not only possible but also a very valuable and effective way to provide support for project management in challenging settings of collaborative and distributed software development.
ISBN, ISSN	ISBN 978-951-38-8177-1 (Soft back ed.) ISBN 978-951-38-8178-8 (URL: http://www.vtt.fi/publications/index.jsp) ISSN-L 2242-119X ISSN 2242-119X (Print) ISSN 2242-1203 (Online)
Date	October 2014
Language	English, Finnish abstract
Pages	101 p. + app. 89 p.
Name of the project	
Commissioned by	
Keywords	Measurements, metrics, global software development, distributed development, project management
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 70

Nimeke	**Mittaamiseen perustuva globaalien ohjelmistokehitysprojektien hallinta**
Tekijä(t)	Maarit Tihinen
Tiivistelmä	Ohjelmistot kehitetään nykyisin globaalisti hajautetuissa, hyvin dynaamisissa ympäristöissä, missä tuotevaatimukset, niiden prioriteetit, tuotekehitysprosessit, työkalut, kehitystyöhön osallistuvat tiimit tai yhteistyökumppanit vaihtuvat tuotteen mukaan. Tällaisten projektien hallinta ja kontrollointi on vaikeaa, joten on tärkeä etsiä uusia menetelmiä projektinhallinnan tueksi. Käytännössä ajantasaisen tilannetiedon saaminen on kriittistä projektin tehokkaan hallinnoinnin kannalta. Toimivien ja tehokkaiden projektinhallinta- ja koordinointimenetelmien avulla on mahdollista jopa välttää ohjelmistokehitysprojektin epäonnistuminen. Ohjelmistokehityksen mittaamisen ja metriikoiden on todettu tuovan useita hyötyjä organisaatiolle. Esimerkiksi kehittyneemmän projektin- ja tuotteenhallinnan kautta on mahdollista päästä tarkempaan arvioon tuotteen markkinoilletuloajasta. Mittaamisen avulla voidaan myös luoda parempaa näkyvyyttä itse tuotekehitysprosessiin ja siten paremmin kontrolloida esim. tuotekehityksen kustannuksia. Niin ikään mittaamisen avulla voidaan parantaa tuotteen laatua ja näin päästä suurempiin myyntimääriin sekä parempaan asiakastyytyväisyyteen. Tyypillisesti yritykset eivät kuitenkaan varaa tarpeeksi aikaa tai resursseja, jotta mittaamiset voitaisiin tehdä tarkoituksenmukaisella tavalla. Päivittäisessä ohjelmistojen kehitystyössä mittaaminen ja metriikat koetaan jopa vieraiksi. Koska globaalissa ohjelmistokehityksessä tiimit, kumppanit ja kehitysympäristöt vaihtuvat projektikohtaisesti, mittausten toteuttaminen on alati haasteellisempaa. Tällaisiin dynaamisiin ympäristöihin ei ole olemassa yhtä tiettyä stabiilia ratkaisua, joka soveltuisi kaikkiin tilanteisiin. Näin ollen globaalin ohjelmistokehityksen mittareiden ja mittaamiskäytäntöjen rakentamisessa on tarkasteltava toimintaympäristöä kokonaisuutena. Tässä työssä on tutkittu ja analysoitu mittaamiseen ja metriikoihin liittyviä ongelmia globaalien ohjelmistokehitysprojektien aikana. Myös kehitystyökaluihin ja -menetelmiin liittyviä haasteita on tunnistettu ja analysoitu. Analysoinnin avulla muodostettiin näkemys esimerkkiratkaisusta, työkaluintegraatiosta. Esimerkin avulla kerättiin tietoa mittaamiseen perustuvasta ohjelmistokehitysprojektien hallinnasta erilaisissa globaaleissa tuotekehitysympäristöissä. Väitöstyössä määritellään dynaamisen mittaamisen käsite, jota tutkittiin kehitetyn työkaluintegraatioesimerkin avulla. Dynaamisen mittaamisen käsite on laaja. Se käsittää mittarit, mittaridatan, tulosten analysoinnin sekä itse mittausprosessin. Mittarit määritellään projektikohtaisesti eri yhteistyötahot huomioiden. Mittaustiedot kerätään eri työkaluista ja tietokannoista, jopa eri yhteistyökumppaneiden tietokannoista. Mittaustulokset muutetaan helposti luettavaan muotoon esimerkiksi visualisoinnin avulla. Toteutettu työkaluintegraatioratkaisu mahdollisti dynaamisen mittaamisen hyötyjen tutkimisen sekä käytännön kokemusten keräämisen yrityksiltä. Tutkimus osoittaa, että dynaaminen mittaaminen tuo hyötyjä globaalien ohjelmistokehitysprojektien hallintaan. Tapaustutkimusten avulla todettiin, että dynaamiset mittaukset mahdollistavat automaattisten ja reaaliaikaisten mittareiden käytön. Mittaustulosten esitystapaan kiinnitettiin erityistä huomiota toteuttamalla visualisoituja helppolukuisia diagrammeja. Tutkimus osoittaa, että automaattisesti tuotetut reaaliaikaiset indikaattorit, joiden mittaustieto on kerätty erilaisista työkaluista ja tietokannoista, jopa eri yhteistyötahojen tietokannoista, ovat erinomainen tapa tuottaa hyödyllistä, luotettavaa ja ajantasaista informaatiota globaalin ohjelmistokehitysprojektin päätöksenteon tueksi.
ISBN, ISSN	ISBN 978-951-38-8177-1 (nid.) ISBN 978-951-38-8178-8 (URL: http://www.vtt.fi/publications/index.jsp) ISSN-L 2242-119X ISSN 2242-119X (Painettu) ISSN 2242-1203 (Verkkojulkaisu)
Julkaisuaika	Lokakuu 2014
Kieli	Englanti, suomenkielinen tiivistelmä
Sivumäärä	101 s. + liitt. 89 s.
Projektin nimi	
Rahoittajat	
Avainsanat	Measurements, metrics, global software development, distributed development, project management
Julkaisija	VTT PL 1000, 02044 VTT, puh. 020 722 111

Measurement-based management of global software development projects

Collaborative, global software development (GSD) has become the norm in software-intensive systems development. Because software products are developed in such dynamic environments, where requirements, priorities, participating sites, development processes and tools, and even partners are continuously changing, project control and management activities are increasingly important. In practice, measurements and metrics provide support for decision making during projects' lifecycle.

In this thesis, the main challenges related to measurements and metrics in GSD projects have been studied and analysed in detail. The thesis provides a definition for *dynamic measurements*, where metrics are defined based on demands of each project's collaboration settings, metrics data is collected and analysed continuously from various tools and databases, and measurement data is analysed and visualised for easy to read format. During the research process, the implemented proof of concept tool integration solution enabled the researcher to study the benefits and industrial experiences of dynamic measurements. The thesis pointed out that automatically produced real-time metrics are a robust and feasible method to produce reliable and up-to-date information for decision making in GSD projects.

The measurement-based management of GSD projects is a very valuable and effective way to provide support for project management in challenging settings of collaborative and distributed software development.

ISBN 978-951-38-8177-1 (Soft back ed.)
ISBN 978-951-38-8178-8 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN-L 2242-119X
ISSN 2242-119X (Print)
ISSN 2242-1203 (Online)