Abstract

Introduction: Large-scale software development requires a substantial amount of coordination. Development and maintenance of code are simultaneously carried out by many individuals and teams, and this code can be integrated and used in many software products. When projects spread through organizational boundaries, such as projects involving multiple companies and autonomous development units, different types of coordination problems appear, which requires suitable coordination mechanisms.

Research objective: This thesis aims at generating new knowledge about coordination challenges in inter-organizational software projects and proposing effective technical and managerial solutions. The thesis addresses three main research questions (RQs):

RQ1. How is the collaboration of technical tasks characterized at organizational level?
RQ2. How do organizational boundaries impact coordination of development activities in distributed software development?
RQ3. How can coordination practices and tools support software development across organizational boundaries?

Research approach: This thesis reports about five studies conducted during four years, focusing on collaboration and coordination of software development activities, such as open source software (OSS) adoption, bug fixing, and duplicated code resolution in different types of inter-organizational projects. The research was performed in various forms, such as systematic literature review, survey, case study and action research. Qualitative data were mainly collected in the form of interviews and documents. Quantitative data were mainly collected from code repositories, issue tracking systems, and mailing lists.

Results: Through a synthesis of contributions from eight papers (MP1-MP8), five contributions connected to the three RQs have been identified:

C1. Identification of challenges and solutions for resolving OSS component integration mismatches in commercial companies
C2. Synthesized impact of geographical and temporal boundaries on team performance and software quality in DSD
C3. A classification framework presenting organizational boundaries that inhibit team coordination activities during software development in DSD
C4. A description of boundary spanners’ capacities and activities in coordinating cross-organization development tasks
C5. Recommendations to management and coordination of development tasks in multiple platform systems

Conclusions: Organizational boundaries inhibit team coordination through collaboration policy, team structure, engineering process, and development practice. Coordination needs across organizational boundaries depend on not only the technical nature of tasks, but also organizational policies. Relying on boundary spanners and cloning mechanisms across configuration management systems, are two effective coordination approaches. Boundary spanners mediate task dependency, status information, handle impacts of global boundaries, and facilitate practice exchanges. Configuration management systems temporize coordination needs resolution, support identifying and resolving duplicate dependencies and facilitate cross boundary meetings. Our research comes up with several recommendations to improve coordination in specific project contexts. Future research in DSD should focus on team coordination in an OSS context, co-existence of competition and collaboration in commercial software ecosystems, and branching and merging practices in multi-platform projects.
Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of philosophiae doctor.

This doctoral work has been performed at the Department of Computer and Information Science, NTNU, Trondheim, under the supervision of Professor Reidar Conradi as the main supervisor and Dr. Carl-Fredrik Sørensen, Dr. Daniela S. Cruzes and Adjunct Professor Torgeir Dingsøyr as co-advisors.

This Ph.D thesis is financed by an internal scholarship from the Department of Computer and Information Science and the Faculty of Information Technology, Mathematics and Electrical Engineering at NTNU. The work has been carried on in four years, including one year of teaching duty.
Acknowledgement

This thesis is financed by an internal scholarship from the Department of Computer and Information Science (IDI) and the Faculty of Information Technology, Mathematics and Electrical Engineering at NTNU, and to some extent by Norwegian Research Council. I would like to acknowledge IDI, NTNU, and the Norwegian Research Council for their financial support.

Next, I would like to thank Professor Reidar Conradi for his supervision during my time as a Ph.D student. I would like to thank Doctor Daniela S. Cruzes for her thorough supervision, and kind guidance in both methodological and technical matters. In the later phase of my Ph.D, I would like to thank Carl-Fredrik Sørensen for his timely advice and kind pressure when needed. Besides, I would like to thank my colleagues at IDI, NTNU for our thoughtful discussions and feedbacks, in particular Tosin, Mohsen, Anca, Eric, Torstein and other people who participated in Forskerfabrikken.

I met a number of researchers outside of my research group who inspired me in many different ways. During my stay at Avaya (New Jersey, USA), I had time to learn about state-of-the-art software development and to write some of the most important papers included in this thesis; therefore I would like to thank Professor Audris Mockus for being my host and my supervisor.

I would like to express my appreciation to my teacher in Blekinge Institute of Technology, Associate Professor Darja Šmite, for her contributive feedbacks on my papers and the Ph.D progress. I would like to thank Professor Jim Herbsleb from Institute for Software Research, Carnegie Mellon University, for thoughtful discussions about the research directions in GSD.

My thanks also go to companies where my data were collected. The open and long-term collaboration with them forms the basis of my work. I would like to thank individuals who have participated in my interviews and surveys and who have given me access to their data repositories.

Finally, I offer heartfelt thanks to my family for their care, support, and encouragement. Special thanks go to my mother Lan Le-Hong and my father Can Nguyen-Chuyen, who have always motivated me, kept me focused and on track during the whole Ph.D.

NTNU, Trondheim, January 15, 2015
Anh Nguyen-Duc
# Table of Contents

Abstract ................................................................................................................................. i  
Preface ................................................................................................................................. iii  
Acknowledgement .................................................................................................................. v  
Table of Contents ................................................................................................................. vii  
List of Tables .......................................................................................................................... xi  
List of Figures ........................................................................................................................ xiii  
Abbreviations ....................................................................................................................... xv  

1. Introduction .......................................................................................................................... 1  
1.1. Thesis Outline ................................................................................................................ 1  
1.2. Problem Outline ............................................................................................................ 2  
1.3. Research Context.......................................................................................................... 3  
1.4. Research Questions ...................................................................................................... 3  
1.5. Study Landscape .......................................................................................................... 7  
1.6. Selected Papers ............................................................................................................ 9  
1.7. Contribution .................................................................................................................. 13  

2. Background ........................................................................................................................ 17  
2.1. Software Engineering (SE) and SE research ............................................................... 17  
2.1.1. Software Engineering .......................................................................................... 17  
2.1.2. Software Engineering Research .......................................................................... 17  
2.2. Distributed Software Development ........................................................................... 20  
2.2.1. Open Source Software Development (OSSD) ....................................................... 20  
2.2.2. Global Software Development (GSD) .................................................................. 23  
2.3. Coordination of Software Development .................................................................... 25  
2.3.1. Software Dependencies ...................................................................................... 25  
2.3.2. Theories of Coordination in Information System (IS) ........................................... 27  
2.3.3. Classification of Coordination Mechanisms ......................................................... 28  
2.3.4. Consequences and Antecedent of Coordination Issues in GSD ......................... 29  
2.4. Theory of Boundary Spanning ...................................................................................... 30  
2.5. Configuration Management System ........................................................................... 32  
2.5.1. Concepts of Configuration Management System .................................................. 32  
2.5.2. Configuration Management Systems in GSD ....................................................... 33  

3. Research Design and Approach ........................................................................................ 35  
3.1. Selection of Research Methods ................................................................................... 35  
3.2. Data Collection .......................................................................................................... 37  
3.2.1. Interview ............................................................................................................. 37  
3.2.2. Data Repository ................................................................................................... 38  
3.3. Data Analysis Approach ............................................................................................. 38  
3.3.1. Qualitative Analysis ............................................................................................ 38  
3.3.2. Quantitative Analysis .......................................................................................... 39  
3.4. Case Study Context Setting ........................................................................................ 40  
3.4.1. OSS Projects ........................................................................................................ 41  
3.4.2. Commercial Projects ............................................................................................ 43  
3.5. Profiles of Companies Participated in Survey ............................................................. 48  

4. Results ............................................................................................................................... 51
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper SP3 - Dispersion, coordination and team performance in GSD</td>
<td>297</td>
</tr>
<tr>
<td>Appendix C: Interview Guideline in Study S2</td>
<td>299</td>
</tr>
<tr>
<td>Appendix D: Conceptual Scheme in Study S3</td>
<td>307</td>
</tr>
<tr>
<td>Appendix E: Interview Guideline in Study S4</td>
<td>311</td>
</tr>
<tr>
<td>Appendix F: Data Extraction Scripts in Study S5</td>
<td>315</td>
</tr>
<tr>
<td>Appendix G: SNA of Spark Projects in Study S5</td>
<td>323</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Research questions and purpose .................................................................................. 4
Table 2: Distribution of contribution in paper ............................................................................ 10
Table 3: Map of questions, studies, and contributions ................................................................. 15
Table 4: Addressed GSD research area ...................................................................................... 25
Table 5: Application of coordination theories in own studies ..................................................... 27
Table 6: Common coordination mechanisms in DSD .................................................................. 31
Table 7: Map of studies, research methods and context ............................................................... 36
Table 8: Project context setting .................................................................................................. 41
Table 9: European software companies ..................................................................................... 49
Table 10: Summary of findings from each study ......................................................................... 52
Table 11: Spearman correlation with issue resolution time (modified from SP1) ......................... 53
Table 12: Global boundaries vs. Project outcomes (modified from MP4) .................................... 54
Table 13: Cross boundaries coordination mechanisms ............................................................... 57
Table 14: Distribution of boundary spanners across different platforms .................................... 58
Table 15: Synthesized answer for RQ1 ...................................................................................... 59
Table 16: Synthesized answer for RQ2 ..................................................................................... 59
Table 17: Synthesized answer for RQ3 ...................................................................................... 61
Table 18: Summaries of contributions and their novelty ............................................................... 63
Table 19: Threats to validity and solution .................................................................................... 69
Table 20: Summary of recommendations .................................................................................... 75
Table 21: Suggestions for future work ......................................................................................... 78
List of Figures

Figure 1: Map of RQs, studies and papers ................................................................. 8
Figure 2: Studies in the context of problem-solution landscape ................................... 9
Figure 3: Investigated SWEBOK areas ........................................................................ 18
Figure 4: GSD Taxonomy (adapted from Darja et al. 2010) ........................................... 22
Figure 5: Elements of dependencies in SE ................................................................. 26
Figure 6: Mintzberg’s coordination framework (Mintzberg 1980) ................................. 27
Figure 7: Distributed VCS and Centralized VCS ......................................................... 34
Figure 8: Data collection timeframe in S4 .................................................................... 37
Figure 9: Suggested steps for conducting thematic analysis (Cruzes and Dybå 2011) .... 39
Figure 10: Social network from mailing list in Wireshark ............................................ 43
Figure 11: Social network analysis issue tracking system in Wireshark ....................... 43
Figure 12: River’s overall architecture ........................................................................ 44
Figure 13: Spark product family as of 9/12 (Hackbarth et al. 2012) .............................. 46
Figure 14: History of code divergence of a codebase in Spark (Hackbarth et al. 2012) .... 47
Figure 15: Organizational boundaries in DSD ............................................................. 55
Figure 16: Fraction of cloning MRs in Spark projects .................................................. 57
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMQP</td>
<td>Advanced Message Queuing Protocol</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkeley Software Distribution</td>
</tr>
<tr>
<td>CAR</td>
<td>Canonical action research</td>
</tr>
<tr>
<td>CBSD</td>
<td>Component-based software development</td>
</tr>
<tr>
<td>CMMi</td>
<td>Capability Maturity Model Integration</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-The-Shelf</td>
</tr>
<tr>
<td>DSD</td>
<td>Distributed software development</td>
</tr>
<tr>
<td>ECOSS</td>
<td>Experience and Challenges with Open Source Software</td>
</tr>
<tr>
<td>ESE</td>
<td>Empirical software engineering</td>
</tr>
<tr>
<td>FP</td>
<td>Function point</td>
</tr>
<tr>
<td>FSF</td>
<td>Free software foundation</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU is Not Unix</td>
</tr>
<tr>
<td>GQM</td>
<td>Goal question metric</td>
</tr>
<tr>
<td>GSD</td>
<td>Global software development</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and communication technology</td>
</tr>
<tr>
<td>IMS</td>
<td>Information management system</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet relay chat</td>
</tr>
<tr>
<td>IS</td>
<td>Information System</td>
</tr>
<tr>
<td>KLOC</td>
<td>Thousand line of code</td>
</tr>
<tr>
<td>MR</td>
<td>Modification request</td>
</tr>
<tr>
<td>NCA</td>
<td>Norwegian Coastal Administration</td>
</tr>
<tr>
<td>OSI</td>
<td>Open source initiative</td>
</tr>
<tr>
<td>OSS</td>
<td>Open source software</td>
</tr>
<tr>
<td>OSSD</td>
<td>Open source software development</td>
</tr>
<tr>
<td>OTS</td>
<td>Off-the-shelf</td>
</tr>
<tr>
<td>PV</td>
<td>Product verification</td>
</tr>
<tr>
<td>RE</td>
<td>Requirement Engineering</td>
</tr>
<tr>
<td>SDN</td>
<td>Software defined networking</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>SIL</td>
<td>System Integration Lab</td>
</tr>
<tr>
<td>SIT</td>
<td>System integration testing</td>
</tr>
<tr>
<td>SLR</td>
<td>Systematic literature review</td>
</tr>
<tr>
<td>SNA</td>
<td>Social network analysis</td>
</tr>
<tr>
<td>SV</td>
<td>System verification</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion</td>
</tr>
<tr>
<td>SWEBOK</td>
<td>Software Engineering Body of Knowledge</td>
</tr>
<tr>
<td>TFS</td>
<td>Team foundation server</td>
</tr>
<tr>
<td>TWIST</td>
<td>Transaction Workflow Innovation Standards Team</td>
</tr>
<tr>
<td>UC</td>
<td>Unified communications</td>
</tr>
<tr>
<td>UoA</td>
<td>Unit of Analysis</td>
</tr>
<tr>
<td>VCS</td>
<td>Version control system</td>
</tr>
</tbody>
</table>
1. Introduction

This chapter provides the outline of this thesis (Section 1.1), research motivation (Section 1.2) and research context (Section 1.3). Three main research questions are proposed to address the research objective (Section 1.4). The chapter includes a brief description of five studies conducted during the Ph.D (Section 1.5) and the resulting eight selected papers (Section 1.6). Finally, a list of contributions to the research community is provided (Section 1.7).

1.1. Thesis Outline

This thesis is structured into six chapters, and several appendixes:

- **Chapter 1** summaries the research questions (RQs), the research context, the research design, and the list of publications and contributions. The detailed discussion of each section is carried on in later chapters.

- **Chapter 2** presents the theoretical background and terminologies that are used in this thesis, including distributed software development, coordination of software development, boundary spanner and configuration management systems. The chapter summarizes relevant theories and state-of-the-art research in collaboration and coordination of software development across global boundaries.

- **Chapter 3** presents the research approaches conducted in this thesis, including research design, the choices of data collection and analysis methods, and description of the study contexts.

- **Chapter 4** gives an overview of the outcomes of this Ph.D research. These results are firstly reported from each individual study. Thereafter, the findings are synthesized to address the three RQs.

- **Chapter 5** evaluates and discusses the research with the respect to the existing literature, research questions, threats to validity, and the Ph.D scope. In addition, it provides a list of recommendations for practitioners.

- **Chapter 6** concludes the thesis and discusses a few possible extensions to this thesis. The implications include input for future research on both distributed software development and coordination of software development.

- **Appendix A** presents the papers that are included in this thesis. **Appendix B** includes abstracts of three supporting papers, which is not included in the thesis. **Appendix C** presents the interview guideline that was used in Study 2. **Appendix D** provides a conceptual scheme of global boundaries, as a result of Study 3. **Appendix E** presents the interview guideline that was used in Study 4. **Appendix F** gives a Perl script that was used to extract data in Study 5. Finally, **Appendix G** presents a social network of investigated projects in Study 5.
1.2. Problem Outline

The increasingly complex and competitive market situation places intense demands on software providers, requiring them to respond to customer needs, and to deliver more functionality and higher quality software in a shorter time than before. These organizations need to use their existing resources as effectively as possible, and they also need to leverage resources all around the world (Herbsleb and Moitra 2001). As a result, distributed software development (DSD) has become a modern paradigm for developing and maintaining software intensive systems. An ACM report shows that 30% of US IT jobs are expected to be offshored by 2015 (ACM Job Migration Task Force 2006). The size of the global market for IT outsourcing is estimated to $270 billion at the end of 2010 and expected to grow at 5%–8% per annum (Oshri, Kotlarsky et al. 2011; Willcocks, Oshri et al. 2011). Distributed software development (DSD) appears in companies of all sizes, and in some estimates, 80% of software projects in 2008 were already globally distributed and predicted to continue growing (Fryer and Gothe 2008).

Adopting DSD strategies, software development organizations aim at shortening time-to-market, technology innovation, increasing operational efficiency and reducing the proximity to customers (Ågerfalk, Fitzgerald et al. 2008). However, there is a number of problems related to economic, social, and technical issues that still need to be solved before the full potential of DSD can be reached. One of the main problems is that team coordination in DSD environments is insufficiently supported. Team coordination refers to timely synchronization and integration of team activities. If coordination is not satisfied or delayed, it can lead to lower team performance and software quality, such as longer task resolution time and more defects (Herbsleb and Mockus 2003, Cataldo and Herbsleb 2011).

In traditional collocated software development projects, poor team coordination was recognized as a critical challenge (Curtis 1989). Nidumolu investigated coordination in software projects and articulated the problem: “How can software development projects be coordinated more effectively in the presence of uncertainty?” (Nidumolu 1995). In a DSD context, team coordination has been even more important and is considered as a crucial success factor for software projects (Herbsleb and Mockus 2003; Noll, Beecham et al. 2010; Šmite, Wohlin et al. 2010). Compared to collocated software projects, a distributed team not only need to handle the uncertainty of coordinated requirements and technical complexity of development tasks, but also need to manage the interdependency between tasks and team members across global boundaries, such as geographical, temporal and cultural boundaries. Recent advancements in configuration management systems and the availability of both low-cost and high-end communication technology have made DSD coordination easier (Cataldo, Wagstrom et al. 2006; Whitehead 2007; Al-Ani and Redmiles 2009; Steinmacher, Chaves et al. 2010; Bendix, Magnusson et al. 2012). However, many companies are still striving to increase the coordination effectiveness of global software development (GSD) projects.

Among the global boundaries, the organizational boundary is an important barrier, yet often neglected in DSD research literature. Reviews of the DSD research revealed the current focus on global boundaries, by pointing out the challenges of collaborating over geographical distances, working in different time zones, and having different perceptions and behavior created by cultural barriers (Noll, Beecham et al. 2010; Šmite, Wohlin et al. 2010). Curtis et al. noted that coordination breakdowns were likely to occur at the organizational boundaries, but that coordination across these boundaries was often extremely important to the project...
success (Curtis 1989). Nagappan et al. showed that some metrics based on the organizational structure have stronger relationship to software quality than the geographical or temporal factors (Nagappan, Murphy et al. 2008). While the study revealed the importance of the organizational boundaries in influencing software quality, the measures of these boundaries are too simplistic to capture different aspects of organizational boundaries. There might be many hidden aspects of organizational boundaries that can have influence on team coordination and project outcomes.

1.3. Research Context

The research was initiated in the context of the Norwegian software industry (Phase 1). According to Statistics Norway (SSB), there are 8,750 software companies in Norway, employing 31,000 people, providing a turnover of 45 billion NOK (SSB 2009). The majority of these companies are small, and only 1,100 of them had five or more employees. Most of the companies focus on consultancy services and provide products mainly to one client. Later, the research scope was extended to software companies that participate in DSD (Phase 2). As this is a global phenomenon, the study samples come not only from Norwegian software projects, but also DSD projects from other countries, such as US, Japan and India.

This Ph.D research has started within the research project ECOSS (Experience and Challenges with Open Source Software). The purpose of ECOSS is to identify methodological challenges and provide solutions for successful adoption of open source software (OSS) in Norwegian, software-intensive companies, across community-company boundaries. In Phase 2, the research focus shifted from a phenomenon of OSS adoption in commercial companies, to a coordination of software development in GSD. Therefore, the main purpose of this thesis is to identify and to understand coordination challenges that arise, and to propose a set of solutions for coordinating tasks across organizational boundaries. The research scope is DSD, which is a combination of both the OSS and GSD context.

1.4. Research Questions

Our research gives a new understanding of the coordination challenges that exist in various types of inter-organizational software development projects, and proposes a technical and managerial approach to improve it. The insufficient amount of context-specific solutions for coordination challenges across organizational boundaries gave rise to the overall problem addressed in this thesis:

| How can coordination of software development activities across organizational boundaries be supported? |

The problem formulation encourages empirical investigations of how software development organizations and communities can successfully realize challenges of working across organizational boundaries and establish effective coordination infrastructures, processes and practices.

Coordination is understood as activities of transferring, integrating and synchronizing information, tasks and software artifacts. As the coordination activities occur among project team members, we would like to study its consequences and antecedences at organizational level. In our context, we study coordination from an inter-disciplinary perspective, as
described detail in Section 2.4.2. The differentiation among concepts of communication, collaboration, and coordination is given in Section 2.2.2.

**Organizational boundaries** are defined as a type of global dispersion dimensions that occur among autonomous organizations (see a more detailed discussion in Section 2.3.1). Example of projects that have organizational boundaries are:

- A commercial company integrates OSS components and utilizes supports from OSS communities (Study 1).
- Commercial companies work together to develop an OSS product (Study 2).
- A client outsources software development tasks to a software vendor (Study 4).
- Competing companies collaborate in a multiple-vendor software development contract (Study 4).
- Autonomous development centers work on a large-scale multi-platform software system (Study 4 and Study 5).

**Software development activities** refer to various types of tasks, such as requirement elicitation, new feature implementation, bug fixing, and code integration (see a more detailed description in Section 2.1). To address this overall problem, the thesis has concretized the problem statement into three main research questions (RQs). The summary of RQs and their purposes are given in Table 1.

### Table 1: Research questions and purpose

<table>
<thead>
<tr>
<th>Questions</th>
<th>Purposes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RQ1: How is collaboration of technical tasks characterized at organizational level?</strong></td>
<td>To explore characteristics of collaboration among organizations for further investigation</td>
</tr>
<tr>
<td>- RQ1.1: How do organizational boundaries impact sociotechnical collaboration in resolving issues in an OSS ecosystem?</td>
<td>To examine the differences in the average OSS issue resolution time between volunteer contributors and commercial contributors</td>
</tr>
<tr>
<td><strong>RQ2: How do organizational boundaries impact coordination of development activities in DSD?</strong></td>
<td>To explore essential properties of organizational boundaries</td>
</tr>
<tr>
<td>- RQ2.1: What is the state-of-the-art about the impact of global boundaries on team coordination and DSD project outcomes?</td>
<td>To understand the state-of-the-art about global dispersion dimensions in DSD and how they relate to project outcomes</td>
</tr>
<tr>
<td>- RQ2.2: Which dimensions of organizational boundaries inhibit coordination of development activities in DSD?</td>
<td>To explore which types of coordination challenges caused by which types of organizational boundaries</td>
</tr>
<tr>
<td><strong>RQ3: How can coordination practices and tools support coordination of development activities across organizational boundaries?</strong></td>
<td>To identify coordination practices that have been successfully adopted in team coordination activities across organizational boundaries</td>
</tr>
<tr>
<td>- RQ3.1: How can boundary spanner roles support coordination of development activities in DSD?</td>
<td>To understand the capacities of boundary spanners to effective coordinate development activities across organizational boundaries</td>
</tr>
<tr>
<td>- RQ3.2: How are configuration management systems used to coordinate development activities in multiple platform software development?</td>
<td>To understand how configuration management systems support coordination activities across organizational boundaries</td>
</tr>
</tbody>
</table>
Each RQ has further been split into two sub-RQs to reflect the particular interest during the different phases of the Ph.D research:

**RQ1: How is the collaboration of technical tasks characterized at the organizational level?**

**RQ1.1: How do organizational boundaries impact sociotechnical collaboration in resolving issues in an OSS ecosystem?**

**RQ1.2: How is OSS component integration performed as a part of collaboration among commercial companies and OSS communities?**

Software products are getting larger and extend beyond the boundary of a single organization. There are many scenarios that a software product is developed by multiple organizations, such as OSS ecosystems and multiple contractors. Collaboration is emphasized by the Software engineering (SE) community as an important success factor. It has been an extensive amount of research focusing on collaboration among developers at the individual and team level (Smite, Wohlin et al. 2010). Differently from the major literature in software engineering, RQ1 seeks to understand how collaboration can be observed at the organizational level (detailed discussion in Section 2.3.1). Driven by the pre-determined research project objectives, the context to study RQ1 is commercial companies that are involved in any type of software development. The term “collaboration” is used to emphasize a focus on working together to create a jointly developed software product between organizations, i.e., commercial companies and OSS communities (detailed discussion in Section 2.3.2). In Phase 1 of the Ph.D, it was not a particularly focus on coordination processes and practices, but more to understand how development activities are done in a collaborative manner in a given context. The collaboration issues were investigated from two viewpoints, namely collaboration within an OSS ecosystem and OSS component integration in commercial companies.

In RQ1.1, the issue resolution process was looked upon as a system of developers and software artifacts. The collaboration among commercial companies in OSS ecosystems is investigated from a socio-technical perspective (see detailed background in Section 2.5). Although many empirical studies in socio-technical collaboration and coordination focus on the project and task levels, only a few studies have been done at the organizational level. By studying the influence of organizational affiliation in the issue resolution process, we can conclude about the existence of organizational boundaries in OSS ecosystems.

In RQ1.2, the OSS component integration process was looked upon as a collaborative work of integrating OSS components among commercial companies, customers, and the OSS community. Literature has investigated OSS components from the perspective of software reuse (detailed discussion in Section 2.3.1). My study focused on the interaction among organizations in finding OSS components and resolving mismatches between the selected components and the customer requirements.

After exploring the collaboration issues among commercial companies and OSS communities, it is straightforward to focus on a narrower topic in a larger research context. Literature reviews and findings from RQ1 suggested that organizational boundaries are meaningful to collaboration between companies in some ways. We focused on coordination activities among companies as it is recognized as a critical success factor in developing large-scale software
products (see detailed discussion in Section 2.4). Consequently, RQ2 seeks to explore essential properties of organizational boundaries, and how they impact coordination activities in DSD. The research context was extended from OSSD (Ph.D Phase 1) to DSD (Ph.D Phase 2) to reflect the state-of-the-practice trends in software engineering research and industry.

**RQ2: How do organizational boundaries impact coordination of development activities in DSD?**

*RQ2.1: What is the state-of-the-art about the impact of global boundaries on team coordination and DSD project outcomes?*

*RQ2.2: Which dimensions of organizational boundaries that inhibit coordination of development activities in DSD?*

Considering an organizational boundary as a type of DSD global boundaries, it is necessary to summarize state-of-the-art SE research about these boundaries in general, and organizational boundary in particular. In RQ2.1, we synthesized different ways of operationalization of global boundaries and their relationship with team coordination, team performance and software quality. The synthesis shows research gaps about coordination across global boundaries and thus gives a basis for further studies in this direction.

In RQ2.2, we identified organizational issues that created team coordination problems and classified them into a framework. Conceptual clarity of organizational boundaries is necessary to understand how these boundaries can be distinguished from other DSD boundaries. Understanding what constitutes the organizational boundaries is also important for sustaining a useful discussion of how to improve it. The question also focuses on the characteristics of organizational boundaries that can directly relate to software development activities, such as code integration and maintenance.

RQ1 and RQ2 aims at giving an understanding about coordination challenges during collaborative software development and organizational boundaries that occur in a DSD context. RQ3 aims at identifying coordination practices that have been successfully adopted in team coordination activities across organizational boundaries. Two coordination mechanisms were of particular interest, namely boundary spanners and configuration management systems. The reason is that they are important mechanisms that emerged by answering RQ2 and they have been little explored in DSD research.

**RQ3: How can coordination practices and tools support software development across organizational boundaries?**

*RQ3.1: How can boundary spanner roles support coordination of development activities in DSD?*

*RQ3.2: How are configuration management systems used to coordinate development activities in multiple platform software development?*

In RQ3.1, we investigated the role of a boundary spanner as a coordination mechanism in DSD. A boundary spanner is a standard organizational design element in product development organizations and is extensively explored in information systems (IS) and engineering management research (see detailed discussion in Section 2.5). Every software project has an individual or a group of people taking responsibility of this role. However, only
a few studies have explicitly investigated this role. Understanding the boundary spanner’s capacities and coordination activities is necessary to encourage more effective practices.

In RQ3.2, we investigated the adoption of configuration management systems as a coordination mechanism. Every modern software project adopts some type of configuration management system. Although many features of the systems are designed to support coordination in DSD, the actual adoption and their effectiveness is unknown (see detailed discussion in Section 2.6). We investigated an MR cloning mechanism in developing software products across multiple platforms, as well as across multiple organizational units.

1.5. Study Landscape

Five studies were conducted over four years, resulting in eleven publications. We have in this thesis, eight publications that directly relate to the RQs. The contributions of these studies contributed to the publications and the RQs are shown in Figure 1.

Study 1 (S1) was conducted in the context of an ongoing research project on understanding software development practices in Norwegian software industry. The study objective was to understand collaboration practices and challenges in OSS ecosystems from a commercial company perspective. We focused on the requirement and component flow across boundaries among commercial companies, client organizations, and OSS communities. The result of S1 was two publications MP2 and MP3, describing state-of-the-practice collaboration among companies in adopting OSS, hence, addressing RQ1.2.

Study 2 (S2) continued the investigation of OSS code adoption and integration from a community perspective. The purpose of S2 was to understand characteristics of collaboration networks among commercial companies in OSS ecosystems. The significant participations of commercial companies in OSS ecosystems might introduce separated segments in OSS projects in term of functionalities focus, productivity and quality of contributed code. This study characterized the bug fixing time among different type of contributors to investigate the existence of organizational boundaries in the OSS ecosystems. The result of S2 was two publications MP1 and SP1, exploring characteristics of company-paid developers in term of OSS project contributions and the relationship to task resolution time. This study contributes to RQ1.2 by investigating whether organizational boundaries matters in OSS projects.

Study 3 (S3) was conducted in a shift to a larger scope, i.e., coordination in DSD. The S3 objective was to classify existing coordination challenges and how these are influenced by global boundaries in the context of DSD. S3 surveyed the GSD research area by systematically reviewing publications on the impact of global boundaries on GSD project outcomes. The results of S3 were two publications MP4 and SP2. One of the results was the list of challenges of organizational boundaries, hence addresses RQ2.1.

While S1, S2, and S3 focus on the problem space, Study 4 (S4) and Study 5 (S5) were conducted in a solution space by characterizing two effective coordination mechanisms, namely the boundary spanner and MR cloning mechanism.

Study (S4) consists of multiple case studies in different DSD contexts, such as offshore insourcing, offshore outsourcing, onshore outsourcing, and OSS projects. In the first phase of Study 4 (S4 Phase 1), we explored characteristics of organizational boundaries in different
scenarios of coordination breakdown, hence addressed RQ2.1. In the second phase of Study S4 (S4 Phase 2), we focused on understanding the role of boundary spanners as a coordination mechanism and how they can be an effective coordination mechanism, which answered RQ3.1.

Study (S5) explored development and coordination practices in a large telecommunication company. The investigated system was a multi-platform messaging engine that involved many functional teams across geographical, temporal, and organizational boundaries. The study scope included (1) exploring development and coordination challenges while working across platform and functional teams, addressing RQ2.2; (2) investigating the adoption of emergent boundary spanners across platforms, addressing RQ3.1, and the use of cloning mechanisms in configuration management system, addressing RQ3.2. The study resulted in two publications, MP7 and MP8.

It should be noticed that these studies were performed according to the transition from the problem domain to the solution domain, and from an OSS context to a commercial company context, as shown in Figure 2. As initiated from OSS research projects, the first two studies S1 and S2 occurred in the context of OSS development and OSS-based software development. In Phase 2 of the Ph.D., I was interested in the phenomenon of coordination in a DSD context. Hence, S3 was performed by reviewing papers in both an OSS and GSD context. S4 consists of multiple case studies, which included both OSS and commercial projects. S5 was a longitudinal case study in a large software development organization. Figure 2 shows the transitions from studies of collaboration and coordination problems, towards a proposal of effective coordination mechanisms and practices.

Figure 1: Map of RQs, studies and papers
1.6. Selected Papers

This section gives a short summary of the eight papers included in this thesis. Together they describe the five main studies we have built our results upon. Their relevance to the thesis is briefly described, as well as our own contribution. The summary of my contribution to each paper is shown in Table 2. The full papers can be found in Appendix A.


(ESEM), Torino, Italy.


1.6.1. Paper MP1 - Impact of Stakeholder Type and Collaboration on Issue Resolution Time in OSS Projects

**Relevance to the thesis:** The objective of this paper was to understand the role of commercial company participants in resolving OSS evolution issues, their collaboration process, and its impact on the OSS issue resolution time. The results suggest that even though a commercial company developer resolves much more issues than a volunteer developer does, there is no difference in issue resolution time between them. Besides, a more important factor that influences the issue resolution time comes from the characteristics of the collaboration network among stakeholders rather than from the individual characteristics. This paper presents our initial findings in S2, and contributes to address RQ1 in characterizing the company-community boundaries in OSS issue resolution time.

**Own contribution:** As the principal author, I was in charge of the study design. I selected OSS projects, conducted data collection and preprocessing. Further, I performed social network analysis (SNA) and statistical analysis on SNA measures. Finally, I was responsible for conducting the literature review. Discussing the results and writing the paper was done collaboratively with two other authors.

1.6.2. Paper MP2 - Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components

**Relevance to the thesis:** The objective of this paper was to understand how functional and non-functional requirement mismatches are handled when OSS code is adopted in commercial projects. We found two common approaches to manage the functional mismatches. Moreover, we found that collaboration with customers enhances functional mismatch resolution while collaboration with the OSS community improves non-functional mismatch resolution. This work suggests OSS code selection should be done iteratively in close collaboration with the stakeholders. This paper presents our findings in S1, and contributes to address RQ1 in characterizing the cooperative approaches on resolving OSS component mismatches from the perspective of commercial companies.

**Table 2: Distribution of contribution in paper**

<table>
<thead>
<tr>
<th>Paper</th>
<th>Study</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S1</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S1</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>S1</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>S3</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S3</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S4</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S4</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S5</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S5</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S5</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>S5</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

*Annotation: M (main responsible), P (participant)*
**Own contribution:** As the main author, I had the main responsibility for writing this paper. It was then reviewed and polished by the other authors. The background section was collaboratively written. Three European institutes jointly performed the study design, interview guideline, and data collection. I was in charge of finding interview candidates and conducting interviews in Norway. Moreover, I was responsible for analyzing the interview material, by coding and re-coding it in NVivo1.

**1.6.3. Paper MP3 - OSS Integration Issues and Community Support: An Integrator Perspective**

**Relevance to the thesis:** The objective of this paper was to understand the common problems that organizations face when integrating OSS components, and the role that the OSS communities play in such integration processes. We found that there is no difference between the way OSS bugs and insource software bugs is located and resolved. We also found that commercial companies managed to deal with their code integration problems by themselves, without requesting specific help from the community. This paper presents our findings in S1, and contributes to address RQ1 in characterizing collaboration issues, means and practices when commercial companies collaborate with OSS community.

**Own contribution:** The study design, interview guideline and data collection were jointly performed by three European institutes. I was in charge of finding interview candidates, conducting interviews in Norway and transcribing them. Claudia Ayala mainly wrote this paper. The discussion of the results was done collaboratively.

**1.6.4. Paper MP4 - The impact of global dispersion on coordination, team performance and software quality – A systematic literature review**

**Relevance to the thesis:** The objective of this paper was to review coordination challenges in GSD literature and synthesize the impact of global boundaries on project outcomes. We selected 42 primary studies reporting the impact of five dispersion dimensions on team performance and software quality. Global boundaries are related to different set of coordination challenges, while organizational boundaries are associated with coordination issues at management level rather than at the technical level. We confirmed the common perception that geographical dispersion has a negative impact on actual team performance and also on manager’s judgment. Temporal dispersion has a positive impact on objective team performance, while it has a negative impact on the perceived team performance. We also found that geographical and temporal dispersion has negative impact on the software quality. The paper contributes to address RQ2 by exploring the organizational dispersion dimensions as part of the GSD dispersion.

**Own contribution:** We performed a Systematic literature review (SLR) to collect relevant studies, thematic analysis to synthesize the extracted data, and vote counting to decide on the impact of global boundaries on project outcomes. I was the main author, responsible for writing all sections. The paper was then reviewed and updated by the two coauthors. I developed the review protocol, searched, and selected relevant studies. The quality of the

1 http://www.qsrinternational.com/products_nvivo.aspx
2 http://qt-project.org
3 https://qpid.apache.org
selected studies was assessed by all authors. I also performed meta-analysis of selected studies and concluded findings.

1.6.5. Paper MP5 - The influence of organizational distance on technical coordination – An exploratory study

Relevance to the thesis: The objective of this paper was to understand aspects of the organizational boundary that inhibits coordination and development activities. We explored a set of effective coordination practices to overcome organizational boundaries. The data were collected from two projects involving four different software development organizations. We found that the variety on collaboration policy, team organization, engineering process, and development practices contributes to extra coordination efforts, insufficient communication, team awareness and mistrust. The study also highlights that coordination practices, such as face-to-face contact, process synchronization, and shared collaborative development are compulsory, but not sufficient for effective team coordination across organizational boundaries. The study contributes mainly to RQ2 and partly addresses RQ3.

Own contribution: As the main author, I was responsible for writing this paper. It was then reviewed and polished by the other authors. I conducted study design, case selection, contacted companies and interviewed relevant stakeholders. Finally, I analyzed the interview material, by coding and re-coding it in NVivo.

1.6.6. Paper MP6 - On the role of boundary spanners as team coordination mechanism in organizationally distributed projects

Relevance to the thesis: The objective of this paper was to understand capabilities and activities of boundary spanning roles in four case studies. The qualitative data were collected from 16 interviews from different types of global software projects. Boundary spanners have common characteristics of a coordinator, such as team member recognition, multiple perspective expertise, decision-making ability, and work time flexibility. Task negotiation, conflict resolution, task information navigation, and boundary object setup are common activities to support team coordination. We also discussed a compound effect of other organizational roles and the impact of context factors on boundary spanner’s activities. This paper contributes to address RQ3 by exploring a type of coordination mechanism in an organizationally distributed context.

Own contribution: As the main author, I was responsible for writing this paper. It was then reviewed and polished by the other authors. I conducted the study design, case selection, contacted companies, and interviewed relevant stakeholders. Finally, I analyzed the interview material, by coding and re-coding it in NVivo.

1.6.7. Paper MP7 - Forking and coordination in multi-platform development: a case study

Relevance to the thesis: The objective of this paper was to understand coordination challenges and practices in a large-scale, multiple platform system. We studied a software product, which has been deployed on more than 25 software/hardware combinations over 10 years, to understand multi-platform development practices. We found the projects using cloned modification requests, cross-platform review meetings, and cross-platform
coordinator’s role as three primary means of coordination. We found that forking code temporarily relieves the coordination needs and is driven by a divergent schedule, market needs, and the organizational policy. Based on our qualitative findings, we proposed quantitative measures of coordination, redundant work, and parallel development. A model of coordination intensity suggests that it is related to the amount of parallel and redundant work. This paper contributes to address RQ3 by investigating coordination mechanisms in large scale multiple platform system.

**Own contribution:** This paper reported a part of an ongoing research project in Avaya. I participated as a relationship worker in the project and had access to necessary stakeholders and data. All authors collectively performed the study design. The study consisted of two phases, a qualitative exploration of multiple platform development and coordination practices, and a quantitative evaluation of coordination mechanisms. In the qualitative phase, I participated in creating the interview guideline, selecting interview candidates, performing interviews and transcribing them. In the quantitative phase, Audris Mockus already had extracted raw data from his previous works. I selected relevant data, linked code files and MRs information and performed regression analysis. All authors jointly did literature review and discussion of results.

**1.6.8. Paper MP8 - A longitudinal case study of coordination in multiple platform development**

**Relevance to the thesis:** The objective of this paper was to further explore the impact of coordination mechanisms on team performance in a large-scale, multiple platform system. We focused on evaluating the effectiveness of coordination mechanisms found in paper MP7. We defined several measures of coordination effort and extents of parallel development and customer-driven software quality. A model of customer-driven software quality suggests that it is positively associated with number of boundary spanners and amount of cloned MRs. This paper contributes to address RQ3 by evaluating coordination mechanisms in large scale multiple platform system.

**Own contribution:** The paper reported a part of an ongoing research project in Avaya. I participated as a relationship worker in the project and had access to necessary stakeholder and data. All authors collectively performed the study design. I selected relevant data, linked code files and MRs information and performed regression analysis. I was also responsible for the literature review section.

**1.7. Contribution**

This Ph.D work is a novel contribution to the DSD research area. In particular, it is a contribution to our understanding of team coordination across organizational boundaries in software development. The contributions consist of five parts:

**C1.Identification of challenges and solutions for resolving OSS component integration mismatches in commercial companies.**

By the summary of qualitative findings from a survey of 25 European software companies, challenges related to requirement negotiation, component selection, bug fixing, and OSS community were identified. Common solutions to mismatches
between selected OSS components and functional/non-functional requirement were also summarized.

C2. Synthesized impact of geographical and temporal boundaries on team performance and software quality in DSD.
By collecting and reviewing empirical papers reporting statistical relationship between measures of global boundaries and measures of team performance and software quality, the overall direction of these relationships are synthesized. The meta-analysis also reveals some context factors that intermediate the relationships.

C3. An empirical framework of organizational boundaries that inhibits team coordination activities during software development in DSD.
By using a list of organizational boundaries found in S3, and a qualitative analysis of interviews in S4, organizational aspects that inhibit team coordination were classified into an empirically based framework. Each element of the framework is linked to coordination and development challenges. Qualitative and quantitative analysis of development challenges in a multiple platform system in S5 validate and complement the framework.

C4. A description of boundary spanners’ capacities and activities in coordinating cross-organization development tasks.
By interviewing and observing project members and boundary spanners in four different DSD contexts, commonalities and differences in boundary spanners’ activities and capacities are gained. Qualitative and quantitative analyses of code commits of emerged boundary spanners in a large organization, validate and complement the list.

C5. Recommendations to management and coordination of development tasks in multiple platform systems.
Qualitative analysis of emergent challenges in multiple platform systems and quantitative analysis of MR and file commit, an insight of how coordination can be supported among diverged codebases. Iterative discussion with developers resulted in a list of recommendation of merging and coordinating diverged codebases in multiple platform systems.

Table 3 describes the relationship between the publications, contributions and research area. A detailed discussion about the contributions and how they contributed to existing the body of knowledge is presented in Section 5.1.
### Table 3: Map of questions, studies, and contributions

<table>
<thead>
<tr>
<th>Questions</th>
<th>Studies</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RQ1: How is the collaboration of technical tasks characterized at the organizational level?</strong></td>
<td>S1, S2, S5</td>
<td>C1, C2</td>
</tr>
<tr>
<td><strong>RQ1.1: How do organizational boundaries impact sociotechnical collaboration in resolving issues in an OSS ecosystem?</strong></td>
<td>S2, S5</td>
<td>C1</td>
</tr>
<tr>
<td><strong>RQ1.2: How is OSS component integration performed as a part of collaboration among commercial companies and OSS communities?</strong></td>
<td>S1</td>
<td>C1</td>
</tr>
<tr>
<td><strong>RQ2: How do organizational boundaries impact coordination of development activities in DSD?</strong></td>
<td>S3, S4, S5</td>
<td>C2, C3</td>
</tr>
<tr>
<td><strong>RQ2.1: What is the state-of-the-art about the impact of global boundaries on team coordination and DSD project outcomes?</strong></td>
<td>S3</td>
<td>C2</td>
</tr>
<tr>
<td><strong>RQ2.2: Which dimensions of organizational boundaries that inhibit coordination of development activities in DSD?</strong></td>
<td>S3, S4, S5</td>
<td>C3</td>
</tr>
<tr>
<td><strong>RQ3: How can coordination practices and tools support software development across organizational boundaries?</strong></td>
<td>S4, S5</td>
<td>C4, C5</td>
</tr>
<tr>
<td><strong>RQ3.1: How can boundary spanner roles support coordination of development activities in DSD?</strong></td>
<td>S4, S5</td>
<td>C4</td>
</tr>
<tr>
<td><strong>RQ3.2: How are configuration management systems used to coordinate development activities in multiple platform software development?</strong></td>
<td>S5</td>
<td>C5</td>
</tr>
</tbody>
</table>
2. Background

Software engineering and research directions in the field are briefly introduced in this Section as a general research foundation for this thesis (Section 2.1). Relevant research in distributed software development is described to provide a common understanding of the research context (Section 2.2). State-of-the-art research in coordination in software development (Section 2.3), a theory of boundary spanning (Section 2.4) and configuration management system (Section 2.5) is provided.

2.1. Software Engineering (SE) and SE research

This section describes the Software Engineering area (Section 2.1.1) and research in Software Engineering (Section 2.1.2).

2.1.1 Software Engineering (SE)

Software engineering researchers often trace the origins of this discipline back to the software crisis (Boehm 1976). The term “software engineering” was firstly mentioned in a NATO conference in 1968, when participants observed “the tendency for there to be a gap, sometimes a rather large gap, between what was hoped for from a complex software system, and what was typically achieved” (Naur and Randell, 1968). SE is different from other kinds of engineering, by the unique, complex and ever changing nature of software development projects (Basili et al., 1986; Mohagheghi, 2004, Kruchten, 2004). According to the SWEBOK (SWEBOK 2001, p. 1-1), SE is defined as a mean to solve to the software crisis:

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

The causes of the software crisis were the rapid increase in computing power, and the increasingly complexity of the problems that could be addressed. As software products become more complex, with a rapidly increase in software size and complexity, deployed on multiple, different types of Operating system (OS) and hardware infrastructures, it was a great need to have better software engineering processes and practices to mitigate the software crisis gap. The direct impact of insufficient management of complexity is a lack of satisfaction with the outcomes, performance, and quality of the delivered software projects and products. In our studies, we focus on improvement of code implementation and maintenance via managerial activities related to coordinating people and adopting configuration management systems, as shown in Figure 3.

2.1.2. Software Engineering Research

Software engineering research has been initialized as a credible mean to inform and influence software practitioners through creating new or revised knowledge in software engineering (Osterweil, 2007). SE research can focus on different types of SE activities with different levels of analysis, as defined in (Sjøberg et al. 2007):
“The development of new, or modification of existing, technologies (process models, methods, techniques, tools or languages) to support SE activities, and the evaluation and comparison of the effect of using such technology in the often very complex interaction of individuals, teams, projects and organizations, and various types of task and software system”

During the last two decades, the emergent trend in SE research has been Empirical Software Engineering (ESE), which gathers and utilizes evidence to advance software engineering methods, processes, techniques, and tools (Basili 1996, Potts 1993; Tichy 1998). The purpose of empirical studies can be to explore a new topic, to describe a phenomenon, or to explain why something happens. For example, an empirical study objective might be to explore the role of a core developer in an OSS communication network, to describe development practices in a multiple platform system, or to evaluate effectiveness of pair programming in agile software teams.

Although ESE research has evolved much, there are still many challenges in solving and influencing practical software development problems, such as research gaps, theories and validation. A detailed list of challenges is shown as:

1. Large gaps in research topics and methods (Hauge 2010): while SE research covers a wide span of topics, the distribution of them is not equal over research topics and methods. (Glass, Vessey et al., 2002; Segal, Grinyer et al., 2005; Höfer and Tichy, 2007). Topics about conceptual models, measurement and prediction, tools, methods, and frameworks are intensively explored (Fenton, 1993; Glass, Vessey et al., 2002; Hansen et al., 2004; Boehm, 2006).

2. Lack of SE theories (Hauge 2010): A lot of the theoretical basis is borrowed from other disciplines, such as decision theory, organization theory, and theory of teamwork (Webb and Jean 1995; Schmidt, Spohrer, et al. 2012; Dingsøyr and Dybå 2012). Although a lot of conceptualization in the SE research, theory is still implicit (Johnson, Ekstedt, et al. 2012), and there is no common agreement on what theories should look like in SE (Hannay, Sjøberg, et al. 2007).
3. **Lack of meaningful empirical validation** (Hauge 2010): SE technical solutions are often proposed without solid empirical evidence about their effectiveness (Fenton, 1993; Tichy, Łukowicz, et al. 1995; Zelkowitz and Wallace, 1998). The importance of empirical studies is gradually recognized in SE research, leading to a rapidly increased amount of SE papers that contain at least some form of empirical studies (Weyuker 2011). However, not all empirical studies are properly conducted, and many of them may not even provide any practically useful findings (Dewayne, Adam et al. 2000).

4. **Limited impact on industry**: many of the SE research problems are loosely connected to SE practitioner’s needs (Potts, 1993). Although much research have been done in an industrial context, they are often limited at the level of describing challenges and offering limited feedbacks for improvement (Wohlin 2013).

5. **Lack of studies on causal relationships**: Empirical studies are dominated by case studies and correlational studies, with very little focus on causal relationship and experiments (Höfer and Tichy 2007). The lack of research into cause and effect seems to be a major weakness in determining the impact of a variable into another, as the values of both of these variables may be determined by other, hidden variables.

6. **Insufficient methodological basis**: Although a lot of empirical research methods exists, adoption of these methods is not always explicit and clearly defined, and the software community is not always reflective about the methods and the method selection. As a result, method selection is not without flaws: the chosen problem might be irrelevant; the proposed solutions might not work in practice; and the context for assessment is not given. Recent guidelines in performing systematic literature review (Kitchenham 2007), case studies (Runeson, Höst, et al. 2012, p. 23), and qualitative data synthesis (Corbin and Strauss 2008, p. 85; Miles and Huberman 2014, p. 12) are a few examples of research guidelines in a SE context. However, there are still a lot of studies in diverse methodological forms.

7. **Lack of studies from multiple perspectives**: many studies only investigate a phenomenon from a particular viewpoint without perspective triangulation. Empirical studies about team performance were often conducted from a particular viewpoint. An example is a study about team coordination in an offshore project which only interviews stakeholders from onshore sites, but is not looking at the opinions of the offshore staff (Gopal and Espinosa 2011). Besides, there is often a mismatch in expectations and perceptions of a project manager and the developers about the project performance.

In general, our studies contribute to SE research not only by focusing on a new and high-impact research topics, but also by adopting thorough research principles:

1. Coordination in global software development is recently gained a lot of interests from academic and industry (see detail in Section 2.2). Improving coordination activities across organizational boundaries is highly of interest, however, there are not many studies addressing this topic.

2. Practitioner-oriented through various types of empirical research in real contexts. Research problems were derived from industrial needs and studied in real life context. We investigated not only coordination challenges in various industrial contexts (S1, S4, S5), but also effective coordination practices (S4, S5). We performed practitioner-driven research by conducting case studies and action research (S4, S5). A longitudinal study was conducted in a long timeframe with close collaboration with internal researchers.
3. Grounded studies from multiple perspectives. We investigated coordination issues from both managers and developers viewpoints (S4), and identified good practices from both client and vendor perspectives (S4). We also explored commercial companies’ development and usage of OSS from company perspective (S1) and OSS community perspective (S2).

2.2. Distributed Software Development

This section describes two of the development contexts used in our studies, namely open source software development (Section 2.2.1) and global software development (Section 2.2.2).

2.2.1. Open Source Software Development (OSSD)

2.2.1.1. Software development processes and practices in OSSD

Open source software (OSS) initiated an ideologically driven software development methodology, where volunteers created communities to develop software, contributing their time and effort with no intention of monetary benefits. An OSS product can be defined as a software/program released with a software license approved by either the Open Source Initiative (OSI) or the Free Software Foundation (FSF). However, the OSS phenomenon is much more than just software products. OSS has a multidisciplinary nature and may be understood as software products, communities, software development processes, release management processes, and business models (Brown and Brook 2002, Gacek and Arief 2004).

As a software development methodology, OSS development (OSSD) has a lot in commonality with commercial software projects. OSS projects are also a type of DSD, since they involve various types of contributors from different locations, working in a common online infrastructure (Scacchi et al. 2006). OSSD is a highly distributed and collaborative activity as development activities are often distributed geographically, with limited direct communication (Nicolas 2003). However, OSS projects are commonly recognized by their informal organization without any pre-assigned governance structure (Bird, Pattison, et al. 2008), volunteer-driven approaches in task assignment (Crowston, Wei, et al. 2012) and contributions, and little or no market/customer pressure for deadlines by customer push or market demand (Scacchi 2007).

Modern OSSD is not a homogeneous phenomenon but follows a common philosophy (Theunissen, Kourie, et al. 2008; Østerlie and Jaccheri 2007; Feller and Fitzgerald 2002, p. 15). There is no common, single OSSD process among OSS projects. There are, however, many common OSS processes and practices observed from successful OSS projects, such as universal, immediate access to all project artifacts, unified collaborative development environment, standards to validate the project, scope decision-making, enabling reuse, release early and often, and peer review of critical code (Robbins 2005). Besides, requirements of new features and bug reports are often circulated in issue tracking systems, such as Jira and Jazz. Development is done via submission of code in configuration management systems. These systems give detailed information about bugs, including information about the status of bugs (open/closed/resolved), who reported them, comments from developers, and an overview of related bugs.
Collaborative OSS development has introduced changes in development processes and practices in commercial software projects. Collaboration among contributors is done via an OSS infrastructure, such as mailing lists, bulletin boards, and threaded messages, allowing participants to describe, study, and question the events that take place in a development project (Scacchi 2007). Communication occurs in an open-access infrastructure, which allow easy browsing of prior discussions and events taking place several years earlier. Many of these practices and processes have been suggested for commercial software development to address some of the challenges of communication, collaboration and coordination across global boundaries (Asundi 2001; Mockus and Herbsleb 2002; Erenkrantz and Taylor 2003). The similarity in coordination infrastructures and practices between OSSD and GSD, suggests a comparative study in both contexts. Implicit assumptions about availability of a development infrastructure, i.e., configuration management systems, and the role of coordinators, have been made throughout study S1, S4 and S5. A detail comparison about coordination between OSS projects and commercial projects is given in paper MP5 and MP6.

2.2.1.2. Commercial participation in OSSD

Governance and development of OSS has been dramatically changed (Santos, Kuk, et al. 2011). The traditional volunteer-based OSS project model has been shifting to a company-sponsored OSS project model, where commercial stakeholders provide efforts beyond voluntary programmers. There has been a tremendous growth the recent years of commercial companies that adopt and participate in OSS. Lakhani et al. report that approximately 40% of developers are paid by their company to contribute to OSS development (Lakhani and Wolf 2005). Bonaccorsi et al. found similar results in a survey of 300 OSS projects hosted on SourceForge, where almost one in three of the projects had one or more companies involved in the development (Bonaccorsi, Lorenzi, et al. 2007).

Wide adoption of OSS has changed the way software companies develop, acquire, use, and commercialize software (Hauge, Ayala, et al. 2010; Ayala, Cruzes, et al. 2011). There are several ways to adopt OSS and correlate them with the benefits of organizations (Hauge, Ayala, et al. 2010):

1. Using OSS CASE tools to facilitate software development, for example OSS collaborative development environments, compilers, control version systems, and modelers such as Eclipse, CVS, and Git.
2. Deploying OSS software such as Linux, MySQL, and Apache Server in the organization’s operational environment.
3. Using software development practices often associated with OSS communities, such as code sharing and peer reviewing within and beyond the organization’s boundary.
4. Integrating OSS components into commercial software products by modifying, extending, or wrapping them—for instance, adopting Hibernate, Google Web Tool Kit, and the Qt framework.
5. Participating in existing OSS product development communities without having decisive control over the product or community—for example, participating in the Linux, Eclipse, and Moodle communities.
6. Providing OSS products and establishing communities to support them, controlling the product development and the community supporting it—for example, organizations providing MySQL, Qt, and JBoss.
The company-community relationships have been explored from strategic perspective (Dahlander and Magnusson 2005, 2008; Capra et al. 2005). Three types of company-community relationships are identified:

a) **Symbiotic**: both OSS communities and commercial company gain benefits from the relationship.

b) **Commensalistic**: commercial companies gain benefits from the relationship but OSS communities are not affected.

c) **Parasitic**: commercial companies gain benefits from the relationship but OSS communities are negatively affected.

Even though there is a large amount of research on the challenges of integrating OSS components in the development of software products (Hauge, Ayala, et al. 2010; Ayala, Cruzes, et al. 2011), the majority of these works investigated the phenomenon in a context of a single case study, which provide very limited information about the real industrial landscape of companies integrating OSS components. Furthermore, the role that the OSS community plays on supporting companies to solve their integration problems, has not been much explored. This research gaps was the motivation for conducting our study S2 and S3. A detailed discussion about novel findings about commercial participation in OSS component integration is given in paper MP2 and MP3.
2.2.2. Global Software Development (GSD)

Almost all OSS projects have development activities through multiple geographical locations, and hence, these projects are cases of global software development (GSD). The term Global Software Development (GSD) implies teams of knowledge workers located in various parts of the globe, developing commercially viable software (Carmel 2001). A global software project has a part of or all software development activities (eliciting requirements, design architectures, developing and maintaining), which occur in a globally dispersed environment. The software is rapidly increasing in size and complexity, and the development teams are also larger and globally dispersed. SE literature presents different reasons for the popularity of the GSD phenomenon in software industry (Sood, 2005; Carmel, 1999; Mockus and Weiss 2001; Prikladnicki, Audy et al., 2003; Vanzin, Ribeiro, et al., 2005):

1. To save development cost: cost is saved by replacing highly paid human resources in an onshore location by a lower-cost human resource in other countries (Ågerfalk, Fitzgerald, et al. 2008). Emerging nations with low software labor costs such as India, Canada, Philippines, Ukraine, Russia, Ireland, China, and Vietnam are popular offshore locations.

2. To reduce operational cost: costs are saved by reducing facility fees such as infrastructure management, human resources, management, payroll, etc. Many maintenance services, such as call centers, have been moved out of USA into third world nations.

3. To reduce the time to market: the time is reduced by adopting ‘follow-the-sun’ strategy (Carmel, 1999). An organization manages resources in multiple time zones, maximizing team productivity by increasing the overall number of work hours during a 24-hour day.

4. To cope with global trends of acquisitions and mergers: nowadays most of the large software companies acquire a company, buy a new product family, or merge with other companies. To remain a viable global player, software companies are increasing global software development by going into merger activities. Due to this trend, software teams from thousands of miles apart are suddenly forced to collaborate and coordinate, sometimes rather reluctantly (Carmel, 1999).

Darja et al. developed an empirically based glossary for classifying four key scenarios in GSD, which are (1) offshore insourcing (using company-internal resources located in a different country), (2) offshore outsourcing (using external third-party resources located in a different country), (3) onshore insourcing (using company-internal resources located in the same country), and (4) onshore outsourcing (using external third-party resources located in the same country). These terminologies are used in our studies to classify the project contexts. Figure 4 shows that combination of GSD type, location, legal entity, and different geographical and temporal distances create even more fine-grained GSD scenarios (Darja et al. 2010). Besides geographical and temporal boundaries, there are many other aspects of global distribution that matters in a GSD software project. Global distribution can be grouped into five dimensions (Nguyen-Duc, Cruzes, et al. 2012):

1. Geographical dispersion occurs when project members are located in different physical places, from where it is not possible for them to perform direct communication since they do not work in the same room (Knobben and Oerlemans 2006; Espinosa, DeLone et al. 2006; Holmström, Conchür, et al. 2006; Šmite, Wohlin et al. 2010; Noll, Beecham, et al. 2011).
2. **Temporal dispersion** occurs when there is a difference in working time among the distributed team members (Knobben and Oerlemans 2006; Espinosa, DeLone et al. 2006; Holmström, Conchúir, et al. 2006; Šmite, Wohlin et al. 2010; Noll, Beecham, et al. 2011).

3. **Work dispersion** refers to a variety of the work environment, such as engineering process, management policy and control (Knobben and Oerlemans 2006; Ågerfalk 2006; M.B. O’Leary and Cummings 2007). Work dispersion is conceptually referred as (1) variations in the engineering process, (2) differences in task and expertise distribution across locations, (3) differences in the work environment and tools and (4) development practices.

4. **Cultural dispersion** is the differences in language, cultural background, and national context of the team members (Knobben and Oerlemans 2006; Espinosa, DeLone et al. 2006; Holmström, Conchúir, et al. 2006; M.B. O’Leary and Cummings 2007, Šmite, Wohlin et al. 2010; Noll, Beecham, et al. 2011). Cultural dispersion is commonly referred as cultural, linguistic, and background differences among team members.

5. **Organizational dispersion** occurs when a team consists of members from different organizations or independent organizational units (Knobben and Oerlemans 2006). Organizational dispersion occurs when (1) a team consists of members from different organizations or independent organizational units; (2) there are differences in the objectives and development strategy among organizational units; (3) there are differences in organizational structure across projects; and (4) existence of different responsibility sharing and team identity.

Although there are some existing classifications of global boundaries, it is still not clear about the ways these boundaries are operationalized and how they are related to measures of coordination and project outcomes. This implies a demand for systematic and quantitative summary of the impact of these boundaries on the project outcomes (as it is done in S4).

It is commonly recognized that communication, collaboration, and coordination of software development across the GSD distribution dimensions are the key to achieve project success (Herbsleb and Moitra 2001; Holmström, Conchúir, et al. 2006; Noll, Beecham, et al. 2010; Prikladnicki, Audy, et al. 2003). In GSD literature, the concepts of coordination, collaboration, and communication are often mixed. In the scope of this work, we differentiate these terms:

- **Communication** is an action of exchanging information between project members, whether formal or informal, synchronous or asynchronous, or occurring in planned or impromptu interaction (Sosa, Eppinger et al. 2002). In an SE context, communication among software engineers can occur via face-to-face meetings, email, comments on a task, chat, online meetings, or telephone. Communication data is basis data element for studying collaboration and coordination (Sosa, Eppinger et al. 2002; Herbsleb and Mockus 2003; Cataldo and Herbsleb 2008).

- **Collaboration** is an action of jointly work by many individuals, teams, and organizations to together achieve a common objective. In SE research, the term is used in a general meaning and covers concepts of communication and coordination (Herbsleb, Mockus, et al. 2006; Jim 2007)

- **Coordination** is an action of orchestrating each task and organizational unit, managing
inter-dependencies among them, so that they all contribute to the overall objective. The detailed theory of coordination is described in Section 2.4.2.

Table 4 presents how collaboration, communication, and coordination are investigated in our studies. Coordination across organizational boundaries is the main research focus, using communication data via issue tracking systems, version control systems and mailing lists.

2.3. Coordination of Software Development

This section presents the theoretical background about software dependencies – a key concept in coordination theory (Section 2.3.1), theory of coordination in Information systems (Section 2.3.2), and state-of-the-art research about coordination in GSD (Section 2.3.3).

2.3.1 Software Dependencies

Generally, the main idea of coordination is to address dependencies, which are the constraints on action in a situation. If there are no dependencies, there is nothing to coordinate. But if a task depends on the result of another task, or if the implementation of a task depends on the expertise of the person that performs the task, dependencies must be managed.

Software complexity can be traced back to the technical dependencies among software components. Traditional software measures have captured structural, logical, and syntactic dependencies by measures of, e.g., cyclomatic complexity, cohesion, and coupling (McCabe 1976; Chidamber and Kemerer 1994). Many software defects are caused by violated dependencies which are not recognized when designing and implementing a software product (Herbsleb, Mockus, et al. 2000). The failure to recognize these dependencies comes from not only the technical nature of the software components, but also the way they are developed. Besides the technical dependencies, the relationship among developers and between developers and software components, can create other types of dependencies, such as technical dependencies, knowledge dependencies, process dependencies, and organizational dependencies:

1. Technical dependencies occur when different developers work on the same module in the same period of time (Wagstrom and Herbsleb 2006). Another type of technical dependency occurs when a change is created that needs to be propagated to different parts of a system; different modules need to be pulled together at integration time; different versions of code files running on different platforms needs to be kept consistently, and API requirements among different modules.

Table 4: Addressed GSD research area

<table>
<thead>
<tr>
<th>Study</th>
<th>Communication</th>
<th>Collaboration</th>
<th>Coordination</th>
<th>Global boundary</th>
<th>Org. boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>-</td>
<td>main topic</td>
<td>-</td>
<td>-</td>
<td>main topic</td>
</tr>
<tr>
<td>S2</td>
<td>support topic</td>
<td>main topic</td>
<td>support topic</td>
<td>-</td>
<td>main topic</td>
</tr>
<tr>
<td>S3</td>
<td>main topic</td>
<td>main topic</td>
<td>main topic</td>
<td>main topic</td>
<td>support topic</td>
</tr>
<tr>
<td>S4</td>
<td>-</td>
<td>-</td>
<td>main topic</td>
<td>support topic</td>
<td>main topic</td>
</tr>
<tr>
<td>S5</td>
<td>-</td>
<td>-</td>
<td>main topic</td>
<td>support topic</td>
<td>main topic</td>
</tr>
</tbody>
</table>

Annotation: (-) not investigated
2. Knowledge dependencies occur when a developer relies on expertise of others (colleagues, customers or testers) to collaboratively accomplish a large technical task (Espinosa, Slaughter, et al. 2007). Besides, developers nearly always rework existing code, modifying it to remove defects, and to add new functionality. Developers always have to work with code written by others when reusing or extending an existing component. Development then becomes a task of aligning your efforts with the work of previous developers. Another type of knowledge dependency is the ability to visualize a software system as a whole. It is common that a developer works on a specific part of the system and does not know about the big picture of the system (Grinter 1996).

3. Process dependencies happen when developers depend on their configuration management systems, the embodiment of their tools and practices; or in collaboration between two teams in different development phases, or different release cycles (Grinter 1996).

4. Organizational dependencies are when a functional team depends on another, such as a dependency between a development team and a system test team. Another example is the dependency between a software development team and a vendor of products, or a tool that the software runs on or is built on. Customers from external organizations are also a type of stakeholders that creates dependencies by their availability to support and to give feedback (Grinter 1996).

When these dependencies are getting large and complex enough, there is a need to handle them by using coordination mechanisms. Addressing the types of dependencies and to what extent they constraint development and collaboration activities, is necessary in a study about coordination (explored in study S4 and S5). In study S5, we explored a new type of technical dependency, which represent a duplication of development task across projects. Figure 5 describes the landscape of different types of software dependencies.

Figure 5: Elements of dependencies in SE
### Table 5: Application of coordination theories in own studies

<table>
<thead>
<tr>
<th>Theoretical basis</th>
<th>Where to use</th>
<th>How to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malone’s inter-disciplinary coordination theory</td>
<td>S1, S5</td>
<td>Basic of formation of developer-issue socio-technical network</td>
</tr>
<tr>
<td>Grinter’s dependencies framework</td>
<td>S4, S5</td>
<td>Framework for understanding the context and coordination needs in DSD projects</td>
</tr>
<tr>
<td>Thompson’s classification</td>
<td>S3, S5</td>
<td>Framework to classify coordination mechanisms in GSD project</td>
</tr>
<tr>
<td>Mintzberg’s framework</td>
<td>S4</td>
<td>Basic to justify the coordination practices</td>
</tr>
</tbody>
</table>

### 2.3.2. Theories of Coordination in Information System (IS)

No single widely accepted definition of coordination exists. Coordination is considered as a key factor determining the organizational structure in classic organizational science. Various organizational level coordination modes, classifications, and theories have been identified (Galbraith, 1977; Malone and Smith, 1987; March and Simon, 1958; Mintzberg, 1980; Thompson, 1967; Van de Ven et al., 1976). Malone is among the first to propose an inter-disciplinary theory of coordination, which took into account ideas from organization theory, economics, management, and computer science (Malone 1988; Malone and Crowston 1994):

> “when multiple actors pursue goals together, they have to do things to organize themselves that a single actor pursuing the same goals would not have to do. We call these extra organizing activities coordination”

While Coordination Theory in its present form may not be suitable for predicting particular outcomes such as coordination effectiveness or project success, nevertheless it is a valuable tool for better understanding the ways in which particular activities or artifacts support coordination in organizational settings (Strode, Huff et al. 2012). In contrast, from a knowledge management perspective, coordination is defined as a way to handle “problems of sharing, integrating, creating, transforming, and transferring knowledge” (Kotlarsky, Fenema et al., 2008). Coordination of work is therefore an important aspect of knowledge management, teamwork, and leadership (Salas et al., 2005).

Adopting the inter-disciplinary theory of coordination, we have accepted the assumptions about the coordination between developers and software artifacts, which can be derived from historical data in configuration management systems, such as file logs, issue reports, and comments on an issue. Socio-technical coordination in a DSD project can be analyzed from a social network perspective (S1 and S5).

![Mintzberg’s coordination framework (Mintzberg 1980)](image-url)
2.3.3. Classification of Coordination Mechanisms

Stakeholders, practices, processes, and activities that are used to manage project dependencies, are coordination mechanisms. There have been several frameworks classifying coordination mechanisms in information systems, software engineering, and organizational science. Table 5 summaries how the theories of coordination were used in our studies. The suitable theories were adopted to classify, explain, and confirm observations in each study.

Thompson classified coordination activities to be either of an organic or mechanism type (Thompson 1967). Organic coordination is the use of horizontal communication means to coordinate activities, such as informal discussions, developer’s comments, and bug reports, while mechanistic coordination is the use of vertical communication means to coordinate activities in a systematic way, such as task organization, task assignment, schedules, plans, project controls and specifications, routine meetings and status checks. Vertical coordination involves communication via supervisors and line managers whereas horizontal coordination occurs via one-to-one communication between individuals in a non-hierarchical relationship.

Mintzberg proposed three ways that coordination mechanisms are set: mutual adjustment, direct supervision, and standardization, as shown in Figure 6 (Mintzberg 1980). Mutual adjustment coordinates team members by the simple process of informal communication. Direct supervision coordination is achieved by having a team member dedicated to supervise the work of others whose work interrelates, by issuing instructions and monitoring their actions, and thus enforcing control in a project. Standardization has four types: standardization of work processes, standardization of outputs, standardization of skills (as well as knowledge), and standardization of norms.

Standardization of work processes is achieved by specifying the work processes of team members whose work interrelates, e.g., by assembling instructions or job descriptions. Standardization of outputs is achieved by specifying the results, e.g., a software architecture document that specify required software modules. Finally, standardization of skills and knowledge is achieved by training the team members to handle different types of tasks, e.g., a software developer, a tester, and a Scrum master.

Kotlarsky categorized coordination mechanisms into organization design, work-based, technology-based, and social mechanisms (Kotlarsky, and van Fenema et al. 2008). Organization design mechanisms include formal structures such as hierarchies, linking points, teams, and direct contacts. Work-based mechanisms involve the specific modularity of tasks to be accomplished. They include project plans, architectural specifications, standards, categorization systems, software prototypes, and design documents. Technology-based mechanisms support coordination by enabling information capturing, processing, storage, and exchange in a centralized or distributed infrastructure, such as electronic calendaring and scheduling, shared databases, and collaborative development infrastructure. Social (inter-personal) mechanisms involve informal communication activities, working relationships, and social cognition among team members.

A downside of empirical studies on socio-technical coordination is the lack of consideration of coordination mechanisms that cannot be traced from data repositories. While appropriate amount of communication (i.e., comments or email) on a coordination-needed technical issue might help to resolve the issue faster, the more fundamental solution might be at
organizational and managerial levels, such as availability of persons who can resolve the issues and the mechanism to inform the issues to the right persons (Šmite, Moe et al. 2008). This implies that investigation in different types of coordination mechanisms might be helpful for resolving socio-technical dependencies. To understand fundamental issues with coordination, it is necessary to realize what coordination mechanisms are available and how they are used in context (as it is done in study S3 and S4).

2.3.4. Consequences and Antecedent of Coordination Issues in GSD

It is commonly perceived that coordination is one of the most difficult and pervasive problems in large-scale systems (Curtis 1988; Walz, Elam, et al. 1993). Nidumolu investigated the impact of project uncertainty, vertical and horizontal coordination mechanisms on project performance in 64 IT projects, finding that horizontal coordination had a positive effect on project performance (Nidumolu 1995). Andres and Zmud studied the impact of task interdependency, coordination strategy, and goal conflict on software development success using an experimental approach, and found that “organic” coordination was generally superior in supporting success under conditions of high interdependency between organizational sub-units (Andres and Zmud, 2001). Coordination is more difficult in a project with a large extent of uncertainty, i.e., development and maintenance of unstable, and confronting non-routine software components (Kraut and Streeter 1995).

Coordination breakdowns occur when coordination needs are not satisfied in a right time and right manner. It is recognized when a decision about coordination of team members leads to development problems. Such breakdowns can cause a variety of problems that differ somewhat depending on the organizational level at which the breakdown occurs. For example, issues at the team level concern mostly the system design and implementation, while issues at the corporate level typically concern product attributes, progress, schedules, and resources (Curtis et al. 1988). However, the understanding about the term “boundary” is fuzzy in many empirical studies. Coordination issues in different levels of organizational boundaries might not be the same and need different approaches to be resolved.

The opposite of coordination breakdown is coordination effectiveness, where appropriate adoption of coordination mechanisms lead to a successful outcome. A variety of project contexts have been studied (e.g., IT projects, software projects, student projects, software teams) and a variety of measure of success have been used (e.g., project satisfaction, productivity, process success, team effectiveness) (Aladwani, 2002; Andres and Zmud, 2001; Kraut 1995; MacCormack et al., 2001; Nidumolu, 1995; Nidumolu and Subramani, 2003). Overall, this body of research has shown that effective coordination is a necessary element of software project success. We assume that coordination effectiveness can be reflected by (1) perception of developer and managers about on time notification (as in study S4), (2) time to collectively resolve an issue (as in study S1), and (3) customer related defects (as in study S5).

Even with a heavy demand for effective coordination mechanisms in traditional collocated software projects, the coordination challenge is amplified when distribution is introduced. Nowadays, it is commonly recognized that the key difference between collocated and global software development is coordination over global boundaries (Herbsleb, 2007). Overall, global boundaries have been perceived as a negative factor impacting team coordination (Herbsleb and Mockus, 2003). Many of the mechanisms coordinating the work in a collocated
setting are absent or disrupted in a distributed project (Herbsleb, 2007). Several consequences of global boundaries in GSE coordination have been identified:

1. **Large and complex dependency network**: many stakeholders from different sites (Nidumolu 1995), which creates a larger dependency and communication network than in collocated projects.

2. **Delay in communication and coordination**: communication and information exchange take longer time in GSD than in collocated projects (Nidumolu 1995, Damian 2007).

3. **Limited choice of coordination mechanism**: distances among sites disable team coordination in face-to-face manner (Darja et al. 2008).

4. **Difficulty in identifying coordination requirements**: identification of people who are technically inter-dependent is difficult (Damian and Zowghi 2003, Cataldo, Bass et al. 2007).

5. **Difficulty in scheduling tasks**: the involvement of many sites across geographical locations and time zones makes scheduling common meetings difficult (Espinosa, Cummings, et al. 2011).

6. **Lack of team awareness**: GSD project team members are often unaware of tasks and activities of other developers from remote sites (Cataldo, Wagstrom, et al. 2006, Damian, Izquierdo, et al. 2007).

7. **Misinterpretation**: team members can misunderstand each other working in different environments and contextual information is not sufficiently given (Holmström and Conchúir 2006, Boden, Avram, et al. 2012).

Previous studies suggest several coordination practices that represent the best practices of the SE literature. Table 6 presents a list of coordination mechanisms and their type (organic or mechanistic). Even though these mechanisms were in place to increase the ability of the teams to successfully perform development tasks, coordination breakdowns still occurred (Cataldo et al., 2007; Smite, Moe, et al. 2008). The root-cause of ineffective adoption of coordination practices might be that (1) usage of coordination mechanisms are not well coordinated among themselves; (2) available coordination mechanisms does not really address the emergent dependencies; and (3) coordination needs are not attempted to be resolved. This suggests that no single set of coordination mechanisms fits all situations. However, commonalities and differences in coordination approaches among different context might be helpful (as it is done in study S4).

### 2.4. Theory of Boundary Spanning

This section presents an important theory, which is used in S4 and S5. IS literature has emphasized the importance of relying on individuals to perform a boundary spanning role (Davenport and Prusak 1998, p. 68; Hargadon and Sulton 1997; Pawlowski and Robey 2004). According to these works, a boundary spanner is viewed as a role linking: “organizational structure to environmental elements, whether by buffering, moderating, or influencing the environments”. Boundary spanners also translate and frame information from one environment to another in an effort to promote coordination (Tushman and Scanlan 1981a, Aldrich and Herker 1977, Brown and Duguid 1998). It is important to govern boundary spanners roles as a mean of cultivating the organizational abilities to deal with the challenges
of managing global boundaries. The boundary spanners might not be the source of new information and ideas, but they gather and disseminate these to other team members.

In SE and IS research, boundary spanning activities is ideally designed to ensure collective performance goals, to capture interactions between an internal team and stakeholders in the embedded environment such as clients, customers, industry experts, and other mutually interdependent teams. Examples of boundary activities are representing the team to outsiders (e.g., updating and seeking feedback from upper management on team progress), searching for information (e.g., contacting subject matter experts for project-related expertise), and coordinating task performance with other external groups (e.g., communicating action plans and delivery deadlines with other interdependent teams (Tushman and Scanlan 1981a,b).

**Table 6: Common coordination mechanisms in DSD**

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
<th>Type O</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronization of main milestones</td>
<td>The customer and vendor teams keep their own development processes and only the main phases and milestones were synchronized (Paasivaara and Lassenius 2003)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Frequent deliveries</td>
<td>Deliverables are integrated and tested in a daily basis; increasing awareness by making ongoing changes to the system available to all the remote teams (Paasivaara and Lassenius 2003)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Establishment of peer-to-peer links</td>
<td>Arrangement of contact points for manager-to-manager and developer-to-developer communication (Paasivaara and Lassenius 2003)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Early relationship building</td>
<td>Facilitate face-to-face meetings among involved stakeholders in the starting phases of a project (Paasivaara and Lassenius 2003)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Early architectural assignment</td>
<td>Assignment of tasks corresponding to architectural modularization; decrease of task dependencies (Cataldo, Bass, et al. 2007)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Shared infrastructure</td>
<td>Having the same set of communication and development tools across sites (Whitehead and Mistrik 2010)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Boundary spanner</td>
<td>Using an unofficial or official role of boundary spanners as a explicit coordination mechanism (Levina and Vaast 2005)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Centralized structure</td>
<td>Centralizing critical decisions and establishing clear paths of communication (Cataldo, Bass, et al. 2007)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Reducing the need to communicate amongst remote teams by having access to detailed design decisions (Cataldo, Bass, et al. 2007)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Configuration management processes</td>
<td>Identifying relationships, manage, control, audit and report on the changes made to the software (Koulinitch and Sheremetov 1998)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Communication tools</td>
<td>Allowing for exchange of information amongst teams when other coordination mechanisms are not sufficient (Sharp and Robinson 2010)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Periodic meetings</td>
<td>Relaying information from remote teams to others (Cataldo, Bass, et al. 2007)</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Notation: O (organic coordination), M(mechanistic coordination)
Boundary spanners can be a formal role, such as team leaders or project manager, who explicitly monitor project development activities, including coordination of developers. But they can also be any of the team members who connect the members of distributed sub-teams in the project network (Di Marco, Taylor et al. 2010; Levina and Vaast 2005). Carmel et al. found an ability of boundary spanner to cross cultural boundary by resolving cultural gaps and translating linguistic information (Carmel and Agarwal 2001). Pawlowski et al. described an ability to facilitate sharing of expertise, integrating communication of the software development team (Pawlowski and Robey 2004). Levina et al. found that boundary spanners-in-practice actually must engage in the boundary spanning by relating practices of one team to practices of another team, and negotiating the meaning and terms of the relationship (Levina and Vaast 2005).

Coordination mechanisms, such as regular meetings, relationship building, mutual adjustment and direct supervision require substantial non-development efforts. Although different types of boundary spanning activities are present in any DSD project, the question is how they are done, by whom they are done.

2.5. Configuration Management System

This section presents some main concepts about configuration management systems (Section 2.5.1) and a brief summary of empirical studies about configuration management systems in GSD (Section 2.5.2).

2.5.1. Concepts of Configuration Management System

A configuration management system is the central infrastructure of modern software development. The configuration of a system is the functional, physical characteristics of a software product and related artifacts, such as implemented hardware, firmware, and product documentation and achieved in a product (Buckley 1996; Conradi and Westfechtel 1998; Asklund and Magnusson 1999; Keyes 2004). As for hardware, configuration management system is responsible for identifying the configuration of a system component at a specific point in time to control changes and to maintain the integrity and traceability of the configuration throughout the system life cycle (P. Bourque and R. Dupuis 2004).

In SE, software configuration managements are responsible for tracing and controlling changes in the software, part of the general configuration management system (Pressman 2000, p. 223). Any infrastructures or set of tools that provides some forms of version control, configuration composition and system modeling, is considered to be a configuration management system (Susan 1991; Estublier 2000). Nowadays, there are several popular configuration management system tools, such as Git, Jazz Source Control, Mercurial, Rational Clearcase and Subversion.

SE research has been noticed that infrastructure of CM needs to come with a well-enforced process over all relevant teams. Without a CM process, different team members can use different versions of software artifacts in different ways without appropriate authority. This leads to a scenario that different version of a code file or bug report can be used in different codebase, causing inconsistency and redundancy over software versions. Successful CM requires a well-defined and institutionalized set of practices and standards, i.e. which set of
One of the core functions of any configuration management system is to coordinate access to a common set of artifacts by multiple developers who are all working on the same project. Configuration management is an approach to visualize and manage technical dependencies. Grinter conducted a naturalistic study of use in an organization of a configuration management tool to coordinate the development of a software product (Grinter 1996). The author found that developers used the configuration management tool routinely to reduce the complexity of coordinating their development efforts. In collocated environments, projects typically have a common configuration management system, a complete development and test environment, good awareness of what other team members are doing and no particular security or network problems (Magnusson and Persson 1999; Fauzi, Bannerman et al. 2010). Any problems or misunderstandings related to the project can be solved quickly through direct communication via face-to-face interaction.

### 2.5.2. Configuration Management Systems in GSD

The situation is different in a DSD environment where communication, coordination and synchronization of development tasks are more significant and problematic. Configuration management system was designed to handle certain aspects of distribution on traditional software projects (Bendix, Magnusson, et al. 2012). Some challenges in DSD can be solved or alleviated by configuration management techniques, such as synchronizing work between distributed sites, spreading team and task information between sites and effective communication. For other types of challenges CM has to be implemented differently or complemented by other development tools, such as task allocation, knowledge management, quality and measurement (Bendix, Magnusson, et al. 2012). While this suggests the limitation of using configuration management across organizational boundary, our studies explored and evaluated how specific features of configuration management can be used as inter-organizational coordination mechanisms (study S5).

Version control systems (VCSs) are specific configuration management systems that focusing on handling versions of code files. There are two general varieties of version control: centralized and distributed. In centralized version control, each developer gets his/her own working copy, but there is just one central repository (as shown in Figure 7a). As soon as one commits, it is possible for co-workers to update and to see the changes. In distributed version control, each user gets his/her own repository and working copy. After code commit, others have no access to the changes until the user push the changes to the central repository (as shown in Figure 7b). Some popular VCSs are Team Foundation Server (centralized), Mercurial (distributed), Git (distributed), and Subversion (centralized). In S5, we investigated code commits from both Git and SVN repositories.

A main issue of a VCS is to handle version conflicts. When writing code in a VCS, a developer might lock the code file before making any modifications. This approach prevents other developers from making concurrent changes and in essence serializes the set of changes to the code file. For a large project involving many developers, especially the ones that implemented distributed VCS, development often occurs in parallel manner, where multiple developers can change the same artifacts at the same time. Usually, VCS is able to merge simultaneous changes by different developers: for each line, the final version is the original
version if neither user edited it, or is the edited version if one of the users edited it. A conflict occurs when two different users make simultaneous, different changes to the same line of a file. In this case, the version control system cannot automatically decide which of the two edits to use. Solution of version conflicts relate to strategies of organizing codebases, coordinating parallel tasks across codebases and the merging parts of the codebases.

Figure 7: Distributed VCS and Centralized VCS
3. Research Design and Approach

In the different research phases, our studies were conducted using different research methods. The reasons to select research methods (Section 3.1), data collection (Section 3.2), and data analysis approach (Section 3.3) are described. Finally, profiles of the companies participating in our case studies (Section 3.4) and surveys are described (Section 3.5).

3.1 Selection of Research Methods

This section describes the research methods and some of the tools for data collection and data analysis used in this thesis. Selection of an empirical approach for the research were motivated by the nature of SE research, characteristics of the research project, and the investigated phenomenon:

1. Software engineering is action-oriented; hence a study is no substitute for action (Pott 1993). Consequently, it is natural to have some form of empirical investigation, which focus on issues that really matter.
2. The research design of the ECOSS project (1) expected an empirical approach to identify a baseline for how the industrial partners leveraged OSS, and (2) conduct process improvement work in collaboration with them. In line with ECOSS, the research methods used in this thesis should allow understanding of real industrial problems and providing evidence-based recommendations to Norwegian software companies and the software industry in general.
3. Coordination and organizational boundaries are human-centric activities and not separable from the project context. Research on organizational boundaries challenges and coordination mechanisms can only be performed in a company environment.

There are many different empirical methods, classified by its research purpose, characteristics of research data, and the research timeframe (see more detailed in Section 2.3). Each of these methods or combinations of them may be applied to almost any research problem. Despite widespread interest in empirical software engineering, there is little guidance on which research methods are suitable to which research problems, and how to choose amongst them. In this thesis, finding appropriate empirical methods was influenced by four factors:

4. The uncertainty of how teams coordinate across organizational boundaries, coordination challenges and collaboration practices motivated an initial explorative and descriptive approach. This was the main motivation for adopting an interview survey and exploratory case studies in papers MP2, MP3, MP5, MP6, and MP7. Case studies were chosen to investigate the contemporary phenomenon of team coordination within its real-life context, (Yin 2003, p. 25).
5. The abundant amount of research about human factors in OSSD and GSD suggests a thorough review before carrying on in-depth research. Moreover, team coordination is also a topic that has been extensively studied in other disciplines, such as IS and Organization Science. Consequently, it is necessary to comprehend what has been done in these fields to give a meaningful contribution. A literature review was performed on team coordination in other disciplines, and a systematic literature review (SLR) on team
coordination challenges and solutions in GSD, as shown in paper MP4.

6. The needs of verifying hypotheses about how team coordination problems impact project outcomes, found from exploratory case studies and literature reviews, motivated confirmatory case studies. Case studies, using both OSS and commercial project data in paper MP1 and MP8, are an appropriate approach due to the accessibility of data and interest of the companies.

7. The ultimate objective of introducing an actual impact in the software industry motivated an action research approach, as used in paper MP8. Baskerville and Wood-Harper suggest that action research, as a research method in the study of human methods, is one of the most scientifically legitimate approaches available (Baskerville and Wood-Harper 1998). Action research is also in line with the basic ideas behind ESE for establishing a fruitful cooperation between research and practice (Dybå, Kitchenham, et al. 2005).

Table 7 describes the research methods and context for each study and paper. In short, four research approaches are used in this thesis: systematic literature review, qualitative survey, case study, and action research:

1. Systematic literature review is defined as “a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest” (Kitchenham 2007; Mark, Barbara, et al. 2008).

2. Case study is defined as “an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin 2003, p. 25).

Table 7: Map of studies, research methods and context

<table>
<thead>
<tr>
<th>Study</th>
<th>Research methods</th>
<th>Data collection</th>
<th>Data analysis</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1: Adoption of OSS in companies</td>
<td>Qualitative survey</td>
<td>Interview</td>
<td>Thematic analysis</td>
<td>25 European software companies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptive statistic</td>
<td></td>
</tr>
<tr>
<td>S2: Company boundary in OSS projects</td>
<td>Confirmative case study</td>
<td>Data repository Interview</td>
<td>Regression analysis</td>
<td>Three OSS projects: Qt, Qpid, Geronimo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3: Organizational boundary in GSD</td>
<td>Systematic literature review</td>
<td>Literature review</td>
<td>Thematic analysis</td>
<td>Literature in GSD, SE and IS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vote counting</td>
<td></td>
</tr>
<tr>
<td>S4: Boundary spanner's coordination activities</td>
<td>Exploratory case study</td>
<td>Interview Documentation Data repository Observation</td>
<td>Thematic analysis</td>
<td>Four software projects: River, Mountain, Ocean, Tree</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Social network analysis</td>
<td></td>
</tr>
<tr>
<td>S5: Coordination in multiple platform development</td>
<td>Case study Action research</td>
<td>Interview Documentation Data repository Observation</td>
<td>Thematic analysis</td>
<td>A large scale US company</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptive statistic Regression analysis</td>
<td></td>
</tr>
</tbody>
</table>
3. Survey is defined as “a system for collecting information from or about people to describe, compare, or explain their knowledge, attitudes, and behavior” (Babbie 1990; Fink 2002).

4. Action research is defined as “a post-positivist social scientific research method, ideally suited to the study of technology in its human context” (Baskerville and WoodHarper 1996; Greenwood and Levin 1998; Davison, Martinsons, et al. 2004).

3.2. Data Collection

This section presents two main data collection approaches, namely interview (Section 3.2.1) and data repository (Section 3.2.2). Other data sources, such as documents, observation notes, and survey responses are described in the individual papers (Appendix A).

3.2.1. Interview

Interview is one main data source in this research. In interview-based data collection, the researcher asks a series of questions to a set of subjects about the areas of interest in the case study. The interview questions are based on the formulated research questions and can be open, i.e., allowing and inviting a broad range of answers and issues from the interviewed subject, or closed offering a limited set of alternative answers. There are three types of interviews, namely unstructured, semi-structured, and fully structured interviews (Robson 2011). In this research, semi-structured interviews were performed, to allow changing the order of interview questions. Appendix C and Appendix E present two interview guidelines used in the studies.

Interview candidates were selected based on a preliminary quantitative analysis, reference from expertise and through a snowball approach. In the different studies, each interview lasted from 30 to 75 minutes, depending on the time availability of the interviewee. The interviews were typically auto-taped (with permission), transcribed, and reviewed by the interviewers. We also sent the transcripts to some interviewees for confirmation. Figure 8 presents the data collection process, which was executed, in the first part of S4. The interviews were performed at different times in the project. Some interview questions were repeated (at a different time, for different companies, and different project roles).

![Figure 8: Data collection timeframe in S4](image-url)
3.2.2. Data Repository

Data repository is another important source of evidence in this research. It includes data from configuration management tools, such as code repositories (SVN and Git), issue tracking systems (Jira), and mailing lists. It is a rich source of historical data of source code, stakeholder interaction, and project performance. Archival data was collected as:

1. Modification requests (MR): A MR is a piece of work (a story, an epic, a task or a bug) that is tracked in an issue tracking system, such as JIRA. In our study, an MR that reports a bug can be initiated by a development team, a product team, a system verification team, or by customers.

2. Commit: A commit is a set of changes to one or more file traced by the Version Control System (VCSs). A commit is normally associated with an MR identifier in the considered projects.

3. Developer information: background information and history of work such as affiliation, bug reports, bug fixes, and list of code commits.

Data extraction is the main task to make use of archival data. There are three steps of extracting archival data:

1. Retrieve the raw data from the database, such as SVN, Git, and JIRA, or search relevant information from the web interfaces of these systems. For example, CVS changes can be obtained via the cvs log command, and commit history of files can be found in SVN and Git log files.

2. Clean and process raw data to remove artefacts introduced by the underlying systems. Verify completeness and validity of extracted attributes by cross-matching information obtained from separate systems. For example, match changes from CVS mail archives, from the CVS log command, or by matching CVS changes to bug reports and identities of contributors.

3. Construct meaningful measures that can be used to assess and model various aspects of software projects.

It should be noted that some of the information that is needed by the researcher may be missing, which means that archival data analysis must be combined with other data collection techniques, e.g., surveys, to obtain missing historical factual data (Flynn, Sakakibara, et al. 1990).

3.3. Data Analysis Approach

This section describes quantitative analysis (Section 3.3.1) and qualitative analysis (Section 3.3.2) that are used in our studies.

3.3.1 Qualitative Analysis

Since case study research is a flexible research method, qualitative data analysis methods are commonly used (Seaman 1999). The basic objective of the analysis is to derive conclusions from the data, keeping a clear chain of evidence. Figure 9 presents four suggested steps for conducting a thematic analysis in ESE (Cruzes and Dybå 2011):
1. *Extract data:* Extract data from the interview transcripts and organize them into meta-fields, such as interviewee background, project background, and explored topics.

2. *Code data:* Identify and code interesting concepts, categories, findings, and results in a systematic fashion across the set of transcripts. Segments were compared from different interviews, but from the same case that were assigned the same code (axial coding). In axial coding, indicators and characteristics for each concept are searched for to define that concept (Miles, Huberman, et al. 2014).

3. *Translate codes into themes:* Translate codes into themes, sub-themes, and higher order themes.

4. *Create a model of higher-order themes:* Explore relationships between themes and create a model of higher-order themes.

### 3.3.2. Quantitative Analysis

Quantitative analyses deal mainly with measuring, comparing and associating numeric data. The main purpose is to characterize data and to reject or not reject a null hypothesis. In our studies, we performed typical analysis, such as descriptive statistics, correlation and regression analysis. We also use social network analysis (SNA) to investigate technical interaction among developers.

**Descriptive statistics:** Statistical data and diagrams such as mean values, standard deviations, histograms, and scatter plots, are used to get an understanding of the data that have been collected. There are some arguments about using simple descriptive statistics in research (Lang and Secic 1997, McGuigan 1995, Altman 1998). In particular, they are concerned that the measures of central tendency and dispersion are often inappropriate. For example means and standard deviations were reported for ordinal variables and heavily skewed interval/ratio variables. In our studies, statistics of skewness and outliers are performed as well. This helps to conduct appropriate pre-processing data activities, such as data cleaning, outlier filtering and data transformation.

**Correlation and regression analysis:** Several hypothesis tests have been performed by either correlation or regression analysis. Correlation analysis was used to find potential variables for association relationship, and regression analysis was used to evaluate the prediction power of the variables of interest:

![Figure 9: Suggested steps for conducting thematic analysis (Cruzes and Dybå 2011)](image-url)
• **Correlation analysis** investigates the extent to which changes in the value of an attribute (such as value of complexity metric in a class) are associated with changes in another attribute (such as number of defects in a class) (Cohen 1977, p. 407). The two most common correlation analyses are the **Pearson correlation coefficient** and the **Spearman rank correlation coefficient**.

• **Regression analysis** is a statistical process for estimating the relationships among variables (Cohen 1977, p. 407). Among the large amount of regression techniques that has been used in software engineering literature, the most common techniques are the **linear regression model** and the **logistic regression model** (Briand, Wüst, et al. 2000).

It has been noticed that neither regression nor correlation analyses can be interpreted as establishing cause-and-effect relationships. They can only indicate how or to what extent variables are associated with each other. The correlation coefficient measures only the degree of linear association between two variables. Any conclusions about a cause-and-effect relationship must be based on the judgment of the analyst. For example, it was adopted a multivariate linear regression model to evaluate the association of measures of human factors and bug resolution time (paper MP1). In another study, a multivariate logistics regression model was used to find if fault proneness is associated with the extent of using coordination mechanisms (paper MP8). In both studies, there was no attempt to draw a causal relationship among investigated variables.

**Social network analysis** (SNA) is used to map and measure relationships and flows between stakeholders, organizations, computer artifacts, and other connected entities. A social network consists of nodes and edges. The nodes are the entities, for instance people, and the edges are the connections between the nodes displaying their relationships, for instance, communication links. Characteristics of the network and its participants are gained by evaluating the location of actors in the network. The location is measured by calculating the centrality of the nodes, which in turn, give insight in the various roles and groupings in the network. Hence, the centrality measure reveals: (1) who are the connectors, leaders or hubs in the network; (2) where there are network clusters, (3) who are in the core of the network, and who are in the periphery. In this thesis, the three most popular centrality measures used are degree, betweenness and closeness centrality (Freeman 2006).

### 3.4. Case Study Context Setting

Table 8 summarizes the context and research methods applied in Study S2, S4, and S5. The projects’ alias, product domain, project size, companies involved, and the global configuration of the project are described. The four OSS projects Qt, Qpid, Geronimo and Wireshark were selected due to the commercial participation, activeness, and convenience of data access. The commercial projects River, Mountain, and Ocean were selected due to their inter-organizational nature, richness and availability of data, and occurrence of coordination issues. It should be noticed that in OSS projects, the project size is the number of developers who had contributed at least once in the project lifetime. In commercial projects, the project size is the number of developers from all involved organizations who have participated in the projects.
3.4.1. OSS Projects

The projects Qt, Qpid and Geronimo were used as datasets to investigate collaboration in Study S2. Project Wireshark has been studied in a longitudinal case, and part of the qualitative data was used in Study S4.

3.4.1.1. Project Qt

Project Qt is a cross-platform development framework developed for desktop and mobile platforms. The framework offers common components such as networking, OpenGL, multimedia, and a widget toolkit. Qt was originally initiated by the Norwegian company Trolltech and was owned by Nokia at the time of this study. Currently, Qt is available under a commercial license, GPL v3, and LGPL v2.

Qt is under open-source governance, involving individual developers and companies working to advance the project. Communication among developers is performed via bug tracking system (JIRA), forum, mailing list, and IRC chat. The bug tracking system provides detailed information about bugs, including information about the status of bugs (open/closed/resolved), who reported them, comments from developers, and an overview of related bugs. Data was collected in a timeframe from Nov-2003 to Dec-2010. At the time of the study, the Qt codebase had 85 active man-months contributed by 1568 developers. These reported and resolved more than 16000 issues. More than 60% of the issues were resolved by Nokia developers.

---

2 http://qt-project.org
3.4.1.2. Project Qpid

Project Qpid is a cross-platform enterprise messaging system developed around the open standard, Advanced Message Queuing Protocol (AMQP). It is implemented in many programming languages such as: C++, C#, Python, Ruby and Java. In 2005, JPMorgan Chase approached other companies to form a working group that included Cisco Systems, IONA Technologies, iMatix, Red Hat, and Transaction Workflow Innovation Standards Team (TWIST). In the same year, JPMorgan Chase partnered with Red Hat to create Apache Qpid, initially in Java and soon after in C++.

Communication among developers is performed via JIRA, forum, mailing list and IRC chat. Data was collected in a timeframe from Sep-2006 to Dec-2010. The Qpid codebase had 51 active months at the time of study, which were contributed by 126 developers, who reported and resolved more than 3000 issues. More than 62% of the issues were resolved by developers from Redhat and JP Morgan.

3.4.1.3. Project Geronimo

Project Geronimo is a server runtime framework that integrates several open source projects to create runtime instances that meet the needs of developers and system administrators. Geronimo is open-source and Apache-licensed. Although the project originated from IBM developers, there are contributions from other commercial companies, such as Mortbay, Intel and IONA. Communication among developers is performed via JIRA, forum, mailing list and IRC chat. Data was collected in a timeframe from Aug-2003 to Dec-2010. Qt codebase had 87 active months when under study, which were contributed by 405 developers, reporting and resolving more than 5500 issues. More than 70% of the issues were resolved by IBM developers.

3.4.1.4. Project Wireshark

Project Wireshark is an open source network package analyzer and has a lot of participation from commercial companies. It is licensed under the GNU General Public License (GPL). The tool offers a rich set of features including: deep inspection of more than a thousand protocols, live monitoring and capturing of data, and offline data analysis. Wireshark is operated through a simple and highly intuitive graphical user interface, and is a cross-platform tool, supporting Windows, Linux, Mac OS X, BSD and Solaris.

Wireshark was initiated by Gerald Combs in the late 1990s. In 2006, Gerald Combs started working for CACE Technologies, and this company became the primary sponsor of Wireshark. In 2010, CACE was acquired by Riverbed Technology, which took over as the primary sponsor for the OSS project. Wireshark is a typical instance of an OSS project. The project uses online infrastructure for development collaboration, and the developers are a mix of company-paid and volunteers. Main contributors to Wireshark are Cisco, Ericsson, Siemens, Netapp, Citrix, and Lucent. Wireshark is also a very successful on-going OSS project, with a high number of contributors and active users, consistently pushing development forward.

---

3 https://qpid.apache.org
4 http://geronimo.apache.org
5 https://www.wireshark.org
Data was collected in the timeframe from Jan-2006 to Dec-2012. There are more than 800 individual developers listed in the contribution list. The Wireshark codebase had 72 active man-months when studied, which were contributed by more than 800 developers, who reported and resolved more than 7800 issues. More than 43% of the issues were resolved by developers from commercial companies. Figure 10 presents an SNA of a company’s communication via a mailing list and Figure 11 presents an SNA of a company’s communication via JIRA.

3.4.2. Commercial Projects

Project River was selected as a follow-up case from the survey participant list in Study S1. Project Mountain was a convenient sample, selected from our contact network in Norway. Project Ocean was selected by introduction of an internal member of the project, located in Vietnam. Project Spark was a part of an on-going action research project in Avaya, where I had spent a 6-months internship.

3.4.2.1. Project River

Project River involved Norwegian Coastal Administration (NCA) and two contractors located in Norway, namely Fundator⁶ and Cap Gemini⁷. Fundator is a small-sized Norwegian software house, located in Trondheim. Cap Gemini AS is a French multinational corporation.

---

⁶ http://fundator.no
⁷ http://www.no.capgemini.com
headquartered in Paris, France. The Cap Gemini office participating in the project is located in Stavanger, Norway. The objective of the project was to develop and maintain a ship management system. The project was more than five years old and included a large amount of legacy code. The main functionality is a messaging service for ships arriving and departing Norwegian ports. The product keeps changing when new functionality is added or the current updated. As shown in Figure 12, the system includes two parts, the thin client (web application) and the thick client (desktop application). Fundator was responsible for the web application, Cap Gemini was responsible for the desktop application. Data access layer and database were jointly developed by both companies.

The project methodology used in the development project was agile. At the beginning of the project, there was little dependency between two teams as both teams only shared a common data access layer and database. When this study was conducted, both teams were performing a joint effort on refactoring the overlapping code base, updating the data model, and developing a new module. There were three aspects of team collaboration:

- **Project owner – external contractor (Fundator/ Cap Gemini):** The communication is mainly via email and telephone. Depending on the project stage, the frequency of email exchange is varying.

- **Fundator – Cap Gemini:** The communication between these two teams was done via email and collaborative development infrastructures. Changes from each team to the shared codebase were made and grouped as batch scripts. Since the shared part was small, the amount of collaboration was around 2-3 weeks per year. Normally, the team leader from Fundator only contacted the bridge engineer (contractor) from the Cap Gemini team.

- **Within the Fundator team:** Since the team is collocated, the face-to-face discussions, both formal and informal were used for team communication. The team leader normally told the team members what to do. Normally, each team member worked on quite different part of the product source code. The team leader took care of the overview of the product, did some testing, automatic testing and was sometimes involved in coding as well.

![Figure 12: River’s overall architecture](image-url)

- **Data access layers (Open source software: Hibernate)**
- **Common parts**
- **Web application system**
- **Desktop application system**
- **Fundator**
- **Capgemini**

44
3.4.2.2. Project Mountain

Project Mountain is a large-scale enterprise search solution, involving the Microsoft headquarter organization and the acquired development center located in Norway. Microsoft\(^8\) headquarter locates in Redmond, Washington, and develops, manufactures, licenses, supports and sells computer software, consumer electronics and personal computers and services. Fast Search & Transfer ASA (FAST) was a Norwegian company based in Oslo, focusing on data search technologies. On April 24, 2008, Microsoft acquired FAST. FAST is now known as Microsoft Development Center Norway.

Project Mountain is a part of an enterprise search product, using a service-oriented architecture development platform geared towards production of searchable indices. The search engine has many API interfaces, and is integrated into Sharepoint and LYNC. The project was more than five years old, involving more than 1500 developers in a global scale. The part of the project that we investigated only related to a team in the US headquarter and two development centers in Norway: Team Everest, Team Dom, and Team Eiger.

Project Mountain involved three roles: developer, tester, and program manager. The program manager is responsible for writing the specification and tracking the development and testing activities. The VCS and bug tracking system were adopted through the whole organization. Common collaboration practices were daily (virtual) meetings, teleconferencing meetings, email, and Team Foundation Server. A few times a year, there was an exchange of team members between USA and Norway. In addition to common collaboration practices for the whole cooperation, the Development Center in Norway retained local collaboration practices, such as communication via Git, informal chats, and frequent visits. Collaboration across sites was a basic need for developing the main system. There was a high amount of technical, temporal and process dependencies among these teams. Each team developed some modules that would be integrated later in the process. The synchronization of tasks was crucial to achieve the overall project milestones and quality of the whole search system.

3.4.2.3. Project Ocean

Project Ocean is a Japanese family registry management system, involving a Vietnamese outsourcing development center, namely FPT\(^9\) and a Japanese customer organization. FPT is a multinational information technology company, based in Hanoi, Vietnam with branches in 14 countries, such as Singapore, Japan, Germany, USA, and Cambodia. The maturity level of the software development process is certified by two quality standards, CMMI 5 and ISO 9001-2000. In the scope of this research, we focused on a project contracted for Hitachi. Hitachi\(^10\) is a Japanese multi-national engineering and electronics conglomerate company headquartered in Chiyoda, Tokyo, Japan. Hitachi had a contract with the Japanese government to develop a large-scale family registry electronic system. The project was more than six years old and its subsystem was outsourced to FPT during the last three years. The project has functionality such as report of births, acknowledgements of paternity, adoptions, disruption of adoptions, deaths, marriages, and divorces of Japanese citizens to their local authority, which compiles such records encompassing all Japanese citizens within their jurisdiction.

---

\(^8\) [http://www.microsoft.com/nb-no/default.aspx](http://www.microsoft.com/nb-no/default.aspx)

\(^9\) [http://fpt-software.com](http://fpt-software.com)

The collaboration was done between FPT and Hitachi. The FPT team adopted a basic agile approach, while the Hitachi site used a tailored waterfall development process. FPT adopted a Management suite, which had a collection of internal tools for project management allowing customers to use remote access to track and manage projects. The tasks were sent from FPT onsite team to offshore team iteratively and integrated to the main system by a joint effort. Coordination between two teams were done via email, site visit and bridge engineer. The common artefacts in FPT were:

- **Timesheet**: for all project members to record their time and effort on tasks
- **Defect management system**: to report, collect and manage internal and beta defects to a software product
- **Fsoft insight**: for project managers to plan, track and monitor projects. Data from timesheet and defect management systems were all linked to Fsoft insight
- **Dashboard**: overview of projects, plans and HR

### 3.4.2.4. Project Spark

Project Spark is a multi-platform engine system developed by Avaya. It is a privately held multi-national corporation which engineers, designs, manufactures, and sells as a global provider of business communications and collaboration systems. The company supplies contact centers, networking (routers, switches and other networking hardware), software defined networking (SDN), unified communications (UC), and video products (integrated hardware and software) services. Development and testing are spread through 10-13 time zones in North America, Europe and Asia.

![Figure 13: Spark product family as of 9/12 (Hackbarth et al. 2012)](image)

11 http://www.avaya.com/usa
A typical development team in Avaya uses Scrum-like development methodologies with a typical 4-week sprint. Each Sprint MR tickets are assigned to a developer/ tester that can complete it in a specific timeframe. If all MRs are finished before the end of the sprint, other MRs will be assigned from the backlog. Code delivery is normally accompanied by an MR ticket. The ticket may be a result of a bug or a new feature. Several mechanistic coordination approaches have been used, such as (1) daily scrum meetings, (2) sprint planning meetings, (3) sprint reviews, (4) backlog review meetings, and (5) Scrum of scrums meetings. A typical project adopts tools: Confluence, JIRA, a version control system (VCS), Project wiki pages, and mailing lists.

A typical test activity is organized in multiple levels of the organization. A SIT team is closely collaborating with a development team, running tests such as unit tests, new feature tests, regression tests, etc. Once SIT Sanity is done – the SIT team will notify that the version is ready for the system verification (SV) team. The SV team is responsible for covering all requirements testing (full SRAD coverage). Once regression tests are executed, SV will notify that the version is ready for alpha release. The product verification (PV) team is responsible for quality insurance of the integration of modules across a product. There is another organizational unit, the so-called SIL (System Integration Lab), which is responsible to do cross product and full stack testing with various Avaya products.

Spark is a 10+ years old, large, complex, real-time software system that are both embedded and standalone products. As a software platform, Spark provides a consistent set of signaling platform functionality to a variety of Avaya telephone product applications. The current evolution of Spark is a client platform that provides signaling manager, session manager, media manager, audio manager, and video manager. Spark engine started in 2005 and has
forked into many different codebases to support different combinations of hardware (Flare, iPad, iPhone, Mac, Samsung devices, 1XC, VDI, ACA, AAC, IPO and LYNC), software (iOS, Android, windows) and communication protocol (SIP, H323, etc), as in Figure 13. Figure 14 describes a history of codebase divergence for 96x1 platform, one of the Spark codebases. 96x1 platform started since May-2010 and has forked into more than ten codebases until Feb-2013.

3.5. Profiles of Companies Participated in Survey

In S2, we surveyed the state of practice OSS adoption in 25 projects from Norway, Sweden, and Spain. Each company is represented by one respondent, who reports an OSS adoption experience in a project. Table 9 shows some of characteristics of a project that was described during interviews with respondents. The project team size ranged from 2-250 people. Especially, company F2 has a four-member project by involving three contributors from an OSS community. Our company pool adopts various software development methodologies, such as ad hoc development, waterfall, iterative development, and agile development, with a prevalence of the agile model in seven projects. The developed product domains range from telecommunication, information systems, web applications, and public sector support. These companies participated in adopting OSS by (1) integrating OSS components into their products, (2) using OSS infrastructure in system operations, (3) and adopting OSS standard protocols in their products.
Table 9: Profiles of survey participants

<table>
<thead>
<tr>
<th>Id</th>
<th>Company size</th>
<th>Domain</th>
<th>Project size</th>
<th>OSS used</th>
<th>Total Effort (hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>170</td>
<td>Defense communication</td>
<td>20-25</td>
<td>JBPM, Jetty, Spring, LogBack, Maven</td>
<td>&gt; 2000</td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>Information system</td>
<td>4</td>
<td>Impact, LPng</td>
<td>480</td>
</tr>
<tr>
<td>F3</td>
<td>3</td>
<td>Information system</td>
<td>2</td>
<td>SolR, Xapian, Twisted, NLTK</td>
<td>12</td>
</tr>
<tr>
<td>F4</td>
<td>350</td>
<td>Embedded systems</td>
<td>18</td>
<td>Linux kernel, MD5 checksum</td>
<td>unknown</td>
</tr>
<tr>
<td>F5</td>
<td>500</td>
<td>Oil and gas industry</td>
<td>2</td>
<td>PDFLib, OpenPyExcel</td>
<td>18</td>
</tr>
<tr>
<td>F6</td>
<td>unknown</td>
<td>Public sector</td>
<td>200</td>
<td>Flex Framework, Spring</td>
<td>unknown</td>
</tr>
<tr>
<td>F7</td>
<td>230</td>
<td>Bank</td>
<td>4</td>
<td>WideShark, CryptoP, ParseX</td>
<td>36</td>
</tr>
<tr>
<td>F8</td>
<td>190</td>
<td>Public sector</td>
<td>20</td>
<td>JBoss, OpenSummer, USD</td>
<td>1000</td>
</tr>
<tr>
<td>F9</td>
<td>6</td>
<td>Finance</td>
<td>2</td>
<td>Soap, Django</td>
<td>3</td>
</tr>
<tr>
<td>F10</td>
<td>4</td>
<td>Public sector</td>
<td>3</td>
<td>SunGridEngine, Cluster FS, Linux Debian, Ganglia</td>
<td>30</td>
</tr>
<tr>
<td>F11</td>
<td>100</td>
<td>Entertainment</td>
<td>3</td>
<td>Apache, MySQL, FFTP tools</td>
<td>7.5</td>
</tr>
<tr>
<td>F12</td>
<td>unknown</td>
<td>Public sector</td>
<td>5</td>
<td>Mantis, Ant, Apache</td>
<td>72</td>
</tr>
<tr>
<td>F13</td>
<td>150</td>
<td>Public sector</td>
<td>6</td>
<td>Jasper Reports, DOJO, Apache, Quark</td>
<td>157</td>
</tr>
<tr>
<td>F14</td>
<td>30</td>
<td>Information system</td>
<td>7</td>
<td>Jenkins, Cucumber, Mercurial</td>
<td>84</td>
</tr>
<tr>
<td>F15</td>
<td>15</td>
<td>Information system</td>
<td>3</td>
<td>Joomla</td>
<td>56</td>
</tr>
<tr>
<td>F16</td>
<td>5</td>
<td>Public sector</td>
<td>3</td>
<td>Zone and Plone</td>
<td>6</td>
</tr>
<tr>
<td>F17</td>
<td>14</td>
<td>Information system</td>
<td>3</td>
<td>Vanish, Engine egg</td>
<td>9</td>
</tr>
<tr>
<td>F18</td>
<td>500</td>
<td>Information system</td>
<td>25</td>
<td>Jasper Reports, Junit, Jmeter, Media Wiki, OpenCSV</td>
<td>900</td>
</tr>
<tr>
<td>F19</td>
<td>2</td>
<td>Public sector</td>
<td>2</td>
<td>RXTX, MySQL, Palcom</td>
<td>36</td>
</tr>
<tr>
<td>F20</td>
<td>&gt;1000</td>
<td>Information system</td>
<td>250</td>
<td>unknown</td>
<td>1000</td>
</tr>
<tr>
<td>F21</td>
<td>11</td>
<td>Energy</td>
<td>2</td>
<td>FPDF, Apache, Stability</td>
<td>20</td>
</tr>
<tr>
<td>F22</td>
<td>2500</td>
<td>Information system</td>
<td>4</td>
<td>Mongo DB</td>
<td>8</td>
</tr>
<tr>
<td>F23</td>
<td>4</td>
<td>Entertainment</td>
<td>10</td>
<td>Apache, MySQL, Suite</td>
<td>24</td>
</tr>
<tr>
<td>F24</td>
<td>1</td>
<td>Public sector</td>
<td>1</td>
<td>UEC, Eucalyptus, NappIt, pfSense, FreeBSD</td>
<td>6</td>
</tr>
<tr>
<td>F25</td>
<td>7</td>
<td>Medical</td>
<td>1</td>
<td>Zone, Plone, Apache, MySql, Ubuntu</td>
<td>3</td>
</tr>
</tbody>
</table>
4. Results

This thesis includes five studies, publishing in eight papers. The studies were conducted during different stages of the Ph.D and are listed in the order in which they were performed. In this chapter, individual findings from each paper are presented (Section 4.1). The relevant findings are then synthesized to address the RQs (Section 4.2).

4.1. Revisit of Study Findings

This section summarizes findings from each study with focus on addressing the research questions presented in Chapter 1. The summary of study findings is shown in Table 10.

4.1.1. Study 1 - OSS integration in commercial companies

Study S1 aimed at investigating the-state-of-practice of the cooperative OSS component integration process in the European software industry. We performed a qualitative survey in commercial companies in Norway, Denmark, Sweden, and Spain. From a pool of 69 commercial companies, we selected 25 companies relevant to OSS adoption and interviewed company representatives. The interview guideline is described in Appendix C. The survey explored several topics, such as requirement engineering, software maintenance, and collaboration practices within OSS communities.

We found that requirement elicitation is a collaborative process, involving customers, software suppliers, and third party vendors/communities. Requirements are elicited iteratively and in parallel with OSS component identification and selection, to select the best-fit set of OSS components that can fulfill the stated requirements. When integrated OSS components mismatch the given requirements, a collaborative approach is used to resolve this conflict. We found that developers either attempt to adapt the selected components, some getting support from the OSS community, some replacing OSS components by commercial-off-the-shelf (COTS) components, and some postponing the requirements by negotiating with customers. Collaboration with customers seems to enhance the functional mismatch resolution, while collaboration with OSS community improves the non-functional mismatch resolution.

It was inquired about the differences among the internal and OSS bugs’ locating/fixing process. Major participants in the survey mentioned the main issues with using OSS component is code ownership, as it is hard to fix bugs in code that someone else wrote, and it requires extra time for code comprehension. OSS components are often adopted as is, without any attempts to locate defects. The survey responses also revealed that OSS components that were selected are often stable components with well-written documentation. With regards to fork OSS codebases, there are always some mismatches between the local fork, which is maintained by internal teams, and the OSS codebase that is updated frequently by OSS contributors. This creates a tradeoff between maintenance cost and degree of code ownership. The advantage of leveraging OSS contributions can be lost when establishing a localized fork of the main codebase.
Table 10: Summary of findings from each study

<table>
<thead>
<tr>
<th>Id</th>
<th>Research objective</th>
<th>Finding</th>
<th>RQs</th>
</tr>
</thead>
</table>
| S1 | To explore the-state-of-practice of the collaborative process of integrating OSS component into commercial software product | • OSS adoption is a collaborative process, involving customers, software suppliers, and third party vendors/communities.  
• Collaboration with customers tends to enhance the functional mismatch resolution, while collaboration with OSS community improves the non-functional mismatch resolution  
• There is limited contribution from commercial companies back to OSS community | RQ1.2 |
| S2 | To examine the differences in OSS issue resolution time between volunteer and commercial contributors | • While volunteer and company-paid assignees participate in OSS projects with different motivation and working approaches, these differences do not have an impact on their issue resolution time | RQ1.1 |
| S3 | To understand concepts of global dispersion dimensions and how they related to coordination and GSD project outcomes | • Different dispersion dimensions are associated with a distinct set of coordination challenges  
• Geographical dispersion has a non-positive impact on the perception of team members on their performance in GSD.  
• Temporal dispersion has a positive impact on the objective team performance while it has a negative impact on the perceived team performance in GSD.  
• Both geographical and temporal dispersion has a negative impact on software quality, at both file and project level in GSD | RQ2.1 |
| S4 | To identify challenges created by the organizational boundaries and the role of boundary spanners as team coordination mechanism. | • Four main groups of organizational challenges, namely collaboration policies, team organization, engineering process, and development practices  
• Four common capacities, which are team member recognition, holding different types of expertise, decision-making ability, and work time flexibility  
• Four types of boundary spanners’ activities that resolve team interdependencies: mediating task dependency, mediating status information, leveraging global boundaries, and mediating practice flow | RQ2.2  
RQ3.1 |
| S5 | To explore how a large software development organization resolve development and coordination problems of a multiple platform system | • Challenges with multiple platform development: growing technical debts, redundant development effort) and two coordination challenges (coordination with procedurally diverged team and coordination across functional units  
• Codebase divergence reasons: schedule constraints, technical variation in the platform, coordination overhead in maintaining a single platform, and organizational differences  
• Four coordination mechanisms: temporizing coordination needs by forking mechanisms, cloned MRs across platforms, a cross platform coordinator role and cross-project MR review meetings | RQ2.2  
RQ3.2 |
The last surveyed topic in this study was to what extent collaboration between commercial companies and the OSS community had been established. We found that there is limited contribution from commercial companies back to OSS community. Besides, companies rarely have dedicated individuals to follow the OSS community. The only situation that communication occurs between an OSS community and companies, are when to locate and seek for bug resolution. Most of the time, the companies just use what is already available on the community portal, or they manage to resolve problems by consulting their colleagues or searching Internet.

4.1.2. Study 2 - Company-company collaboration in OSS bug fixing

Study S2 aimed at understanding characteristics of collaboration networks that are related to organizational identity in an OSS bug fixing process. Particularly, we examined the differences in the average amount of resolved issues and issue resolution time between volunteer contributors and commercial contributors. We extracted contributor comments and bug reports from three OSS projects (as described in Section 3.4.1). We performed statistical tests, correlation analysis (as in Paper MP1) and regression analysis (as in Paper SP1) on measures of collaboration attributes and developer’s affiliation.

The study found that while volunteer and company-paid assignees participate in OSS projects with different motivation and working approaches, these differences do not have an impact on their issue resolution time. The analysis also showed that the high level of collaboration in an issue, such as a high number of messages exchanged and a high number of involved stakeholders, is correlated with a longer resolution time. This may be due to the complexity of the task that relates to other issues or software modules; or that poor quality of an issue description leads to demands of explanation and discussion. Assignee experience is more useful than reporter experience in predicting issue resolution time in a regression model, as shown in Table 11. That is, the developer tends to resolve issues faster when he or she has resolved many issues before. Moreover, past performance of a developer in resolving bugs is also useful to predict time to fix a current issue. The analysis also confirmed previous findings, which stated that issues involving more developers and having more comments would take longer time to resolve.

4.1.3. Study 3 - Impact of global boundaries on GSD project outcomes

Study S3 summaries how dispersion dimensions are defined and measured, coordination issues attached to each dimensions, and how they together impact team performance and software quality. We selected 45 high quality papers from a pool of 11222 GSD papers. The papers were then synthesized by narrative analysis and vote counting.

Table 11: Spearman correlation with issue resolution time (modified from SP1)

<table>
<thead>
<tr>
<th>Variables</th>
<th>Project Qt</th>
<th>Project Qpid</th>
<th>Project Geronimo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug description length</td>
<td>-0.123**</td>
<td>0.065**</td>
<td>0.118**</td>
</tr>
<tr>
<td>Bug priority</td>
<td>-0.157**</td>
<td>0.021</td>
<td>-0.021</td>
</tr>
<tr>
<td>Experience of Reporter</td>
<td>0.372**</td>
<td>0.222**</td>
<td>-0.113**</td>
</tr>
<tr>
<td>Experience of Assignee</td>
<td>-0.186**</td>
<td>-0.021</td>
<td></td>
</tr>
<tr>
<td>Number of Comment</td>
<td>0.008*</td>
<td>0.243**</td>
<td>0.416**</td>
</tr>
<tr>
<td>Number of Stakeholder</td>
<td>0.123**</td>
<td>0.309**</td>
<td>0.303**</td>
</tr>
<tr>
<td>Average past resolution time</td>
<td>0.799**</td>
<td>0.284**</td>
<td>0.222**</td>
</tr>
</tbody>
</table>
We identified five global dispersion dimensions: geographical, temporal, cultural, work process, and organizational boundaries. These dispersion dimensions are consistently conceptualized, but quantified in many different ways. Moreover, different dispersion dimensions are associated with a distinct set of coordination challenges.

We also synthesized how the geographical and temporal dispersion dimensions impact team performance and software quality. With regards to team performance, a distributed task takes longer time to communicate and resolve than a collocated task does in GSD. Geographical dispersion has a non-positive impact on the perception of team members on their performance in GSD, as shown in Table 12. Temporal dispersion has a positive impact on the objective team performance while it has a negative impact on the perceived team performance in GSD. With regards to software quality, both the geographical and temporal dispersion have a negative impact on software quality, at both file and project level in GSD. Work process dispersion also seems to have a negative impact on software quality. The synthesized impact scheme is given in Appendix D. One of the suggestions for future GSD research is to explore the nature of the organizational and work process boundaries in GSD projects, and to investigate their impact on project outcomes.

### 4.1.4. Study 4 - Organizational boundary and boundary spanner

Study S4 investigated challenges that are introduced by the organizational boundaries and possible coordination solutions in inter-organizational contexts. The study was conducted in two phases: Phase 1 explored the challenges with team coordination in inter-organizational context and Phase 2 investigated effective team coordination mechanisms. We performed case studies in GSD projects, involving eight organizations (as described in Section 3.4.2). Data, which was mainly collected from interviews, observations, and documents, were analyzed by thematic analysis.

#### 4.1.4.1. Organizational boundary challenges

In Phase 1, we identified four main groups of organizational challenges, namely collaboration policies, team organization, engineering process, and development practices, as shown in Figure 15. Firstly, we observed a direct impact from a high-level governance policy on the coordination of technical tasks. In Project River, coordination is inhibited by vendor governance policy and competition attitude:

“...so we are in the competition with [Team A1]. Maybe [Team A1] want to do it too so we need to discuss and make argument about why should we do it instead. This is absolutely one part of the competition. But we also want to look good compared to [Team A1]. And we know they want to do the same ...” (Paper MP5)
In Project Mountain, the high-level management policy clashed on the autonomy of each development center:

“We had so much to do that we tried to push the tasks in the borderline between team. We tried to push it to other team …” (Paper MP5)

Secondly, organizing team structure according to product architecture is not a straightforward task in an inter-organizational context. Alignment of communication network and code submission structure is a challenge, as working on a common set of branches and committing code in a shared repository, require a large amount of coordination. A question is, i.e:

“... when the developers check in (to the main branch), should they be free to check in any time or what kind of testing we need to do before checking in? ...” (Paper MP5)

Thirdly, we found that differences on development methodology, such as sprint iterations and team priority, hinder frequency of communication and ability to synchronize tasks across organizational boundary. Even in the same organization, coordination issues introduced by procedural differences should be expected. Depending on the coordination needs, this procedural commonality should allow synchronization activities across teams without extra resources and efforts.

Fourthly, we confirmed that the variation on coding and testing practices introduces a negative perception on the work of others and hinders the establishment of a common understanding (Šmite, Moe et al. 2008, Moe and Šmite 2008). Differences on testing and coding practices can introduce difficulties in establishing a common technical understanding. However, it was also observed coordination problems in a homogeneous development and collaboration infrastructure.

4.1.4.2. Boundary spanner as a coordination mechanism

Phase 2 of the study identified several coordination mechanisms that are effective for managing cross-organization dependencies. Especially, an important coordination mechanism to manage information flow across organization was examined. This mechanism is the *boundary spanner*.

Boundary spanners are individuals who obtain the role of linking the internal networks of the organization with external sources of knowledge and information. Two research topics were
covered: (1) the capacities of boundary spanners in a coordinator role, and (2) the approaches that boundary spanners use to handle dependencies in distributed software projects.

With regards to the boundary spanner’s coordinator role, we identified four common capacities. These are team member recognition, holding different types of expertise, decision-making ability, and work time flexibility. Firstly, boundary spanners were recognized by not only internal team members, but also external stakeholders as an interface to communicate with the other project stakeholders. For a customer, boundary spanners were considered as a source of information and also a target for giving feedback and negotiating tasks. Secondly, boundary spanners have multiple domain knowledge, making it possible to communicate and translate information among different worlds:

“One capacity of a bridge engineer is to be keen on the technology and the development framework, but also be fast on learning business domain and requirement from customers ... As part of my task, I also checked deliverables from [Team Behring] before submitting into customer main branch.” (Paper MP6)

Thirdly, boundary spanners need to be able to make decisions, such as how to leverage boundary distances and to assign tasks to developers and to organize a team. This is an important ability because technical coordination requires actual influence on activities of the relevant stakeholders, such as developers, testers, and customers. Fourthly, as a special position that stands across multiple boundaries, including the temporal and cultural boundary, a boundary spanner needs to be flexible in working time and place.

With regards to a boundary spanner’s coordination approaches, we identified four types of activities that resolve team interdependencies. These are mediating task dependency, mediating status information, leveraging global boundaries, and mediating practice flow. Mediating task dependency is the most frequently mentioned concept, which includes four sub-concepts: comprehending, transferring, negotiating tasks, and integrating deliverables. Mediating status information consists of three sub-concepts: liaising, transporting information, and hiding information. Leveraging global boundaries, such as the geographical boundary, temporal boundary, and cultural boundary is also a part of activities of a boundary spanner. Mediating practice flow is the least mentioned concept from the data transcripts.

### 4.1.5. Study 5 - Forking and coordination in multiple platform development

Study S5 aimed at understanding how a large software-intensive organization resolves development and coordination problems in multi-platform software development. We performed a longitudinal case study of a large scale multiple platform engine system. Data was collected from interviews, documents, and data archives during 15 months. Thematic analysis was used to analyze qualitative data while regression analysis was used to investigate quantitative data.

Firstly, we explored reasons that lead to divergent codebases, which were summarized into four main categories: schedule constraints, technical variation in the platform, coordination overhead in maintaining a single platform, and organizational differences. A fork occurs when a significant amount of new features are introduced to the project, or the project gets into the delivery phase when there are still some functions under rapid changes. In case two projects share the same code base, cross-project coordination is required to manage the impact of
changes to the code base on each project. When the coordination amount becomes too large, the codebase needs to be separated to reduce coordination requirements. There is also a common pattern to fork a project according to the organization structure, as stated:

“... it (coordination efforts) has been mostly on ipad and windows team, because we came from the same team and we were working very closely together on the same area . . . after that, we have been divided into different teams, as the managers were divided into the pure desktop and windows team and the purely mobile teams ...the organization split in the different ways from ...” (Paper MP8)

Secondly, we investigated challenges of working with divergent codebases. There are two development challenges (growing technical debt and redundant development effort) and two coordination challenges (coordination with procedurally diverged team, and coordination across functional units). In the multi-platform project, a bug fix or new feature often needs to be implemented in different versions across different platforms. Changes in one platform that need to be applied to another platform are often delayed or not implemented, and hence, lead to technical debt. Besides, developers responsible for one Spark platform were often unaware of many changes in other Spark platforms. Consequently, features and bug fixes are implemented independently for each platform, resulting in development that is unintentionally duplicated. Moreover, divergent codebases induce redundant test efforts because each platform’s test team independently tests similar features or functionality from the different platforms.

Table 13: Cross boundaries coordination mechanisms

<table>
<thead>
<tr>
<th>Coordination mechanism</th>
<th>Function</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross platform coordinator</td>
<td>Mediate task dependencies, information flow, manage global boundaries and mediate practice flow</td>
<td>Knowledge silo, Recognition from manager</td>
</tr>
<tr>
<td>Cross platform fork</td>
<td>Temporize coordination needs across platforms</td>
<td>Technical debt, Redundant work</td>
</tr>
<tr>
<td>Cloning issues via Jira</td>
<td>Propagation of duplicated tasks to relevant developers</td>
<td>Manual detection effort</td>
</tr>
<tr>
<td>Cross platform review meeting</td>
<td>Maintain awareness common issues cross platform</td>
<td>Formal coordination effort</td>
</tr>
</tbody>
</table>

Figure 16: Fraction of cloning MRs in Spark projects
Thirdly, we identified effective coordination mechanisms across platforms. The qualitative data revealed many coordination mechanisms such as: automatic detection of build failure, continuous building, dashboard, wiki, Jira, and mailing lists. While these mechanisms have been extensively mentioned in SE literature, the study would like to focus on the three approaches that are especially interesting in the context of a multi-platform system. Many projects deployed at least two of these approaches: temporizing coordination needs by forking mechanisms, cloned MRs across platforms, a cross-platform coordinator role, and cross-project MR review meetings. Table 13 presents the main functionalities of these mechanisms and also challenges attached to them.

Cloned MRs across platforms where a cloned MR is a replica of the original MR, contains the same information stored in the original issue, such as Summary, Affected Versions and Components. The cloned MR is typically linked to the original MR. Formal defect tracking systems are used as standard practice in all Spark-related development teams. We found a significant use of cloned MR across Spark projects, as shown in Figure 16.

A cross-platform coordinator is a technical role, who addresses cross-platform issues, an important role to maintain knowledge flow among projects. The study identified two types of coordinators, formally assigned coordinators and emerged ones. Formal coordinators might be a designated team consisting of developers from the different platforms, resolving issues of common concern, such as changes in tools or processes, buyback planning, and dependencies between projects. Emerged coordinators are developers that work on many platforms and hence, they also gained a lot of cross-platform knowledge. These developers are not formally assigned to any coordination activity, but often contribute much by identifying, communicating, and resolving defects across platforms. Table 14 describes statistics of portion of developers that contributed to a certain number of different platforms.

Cross-project MR review meetings include project managers, technical managers, and lead developers from each participating project. At each meeting, participants review requests from each participating team, review and create tickets, and make sure team members are aware of what is coming to them. Cross-project MR meetings help creating a culture of regular informal communication to clarify conflict issues, cloning issues, and other issues.

4.2. Addressing the Research Questions

Individual findings from each study are synthesized according to the three main RQs. We used a thematic analysis on texts that describe the results of the studies to answer RQs. It should be noticed that not all findings from the studies are presented in the answers. The answers are described for RQ1 (Section 4.2.1), RQ2 (Section 4.2.2), and RQ3 (Section 4.2.3).

<table>
<thead>
<tr>
<th># spanned proj.</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min (%)</td>
<td>16,4</td>
<td>12,6</td>
<td>11,1</td>
<td>9,5</td>
<td>8,6</td>
<td>4,7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Quarter 1st (%)</td>
<td>72,6</td>
<td>47,5</td>
<td>42,6</td>
<td>35,05</td>
<td>25,65</td>
<td>16,55</td>
<td>1,95</td>
<td>0,7</td>
<td>0</td>
</tr>
<tr>
<td>Median (%)</td>
<td>90,5</td>
<td>78,6</td>
<td>72,1</td>
<td>67,3</td>
<td>55</td>
<td>25,4</td>
<td>5,2</td>
<td>1,9</td>
<td>0</td>
</tr>
<tr>
<td>Quarter 3rd (%)</td>
<td>95,85</td>
<td>91,15</td>
<td>84,2</td>
<td>78,1</td>
<td>68,95</td>
<td>42,75</td>
<td>13,3</td>
<td>3,6</td>
<td>0</td>
</tr>
<tr>
<td>Max (%)</td>
<td>100</td>
<td>98,6</td>
<td>95,9</td>
<td>93,2</td>
<td>88,1</td>
<td>75</td>
<td>66,7</td>
<td>9,4</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 15: Synthesized answer for RQ1

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Boundary</th>
<th>Study</th>
<th>Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: How is the collaboration of technical tasks characterized at the organizational level?</td>
<td>There is no significant difference introduced by organizational boundaries in issue resolving process</td>
<td>OSS community vs. commercial companies</td>
<td>S1, S2</td>
<td>Issue tracking system, Mailing list</td>
</tr>
<tr>
<td></td>
<td>There is a limited technical collaboration across organizational boundaries in an OSS adoption process</td>
<td>Commercial companies vs. OSS community</td>
<td>S2</td>
<td>Interview</td>
</tr>
<tr>
<td></td>
<td>There is a large amount of coordination needs across organizational boundary in large-scale software systems</td>
<td>Development center vs. headquarter Vendor vs. client</td>
<td>S5</td>
<td>Interview</td>
</tr>
</tbody>
</table>

4.2.1. RQ1: How is the collaboration of technical tasks characterized at organizational level?

In general, the occurrence of organizational boundaries and their impact on development and coordination activities varies in different DSD contexts. Thematic analysis of findings from Studies S1, S2, and S5 describes features of technical collaboration among autonomous organizations in three different contexts, OSS ecosystems, OSS based software projects, and a distributed multi-platform software project:

1. OSS ecosystems: there are no significant differences introduced by organizational boundaries in issue resolving process. It seems that the organizational boundary does not impact the issue resolution process in OSS projects, but rather characteristics of individual contributors, such as developer’s experience and amount of communication.

2. OSS based software projects: there is limited technical collaboration across organizational boundaries in an OSS adoption process. Collaboration with customers and the OSS community tends to improve the mismatch resolution of functional and quality requirements. However, there is little contribution of neither code, bug reports, nor development practices from commercial companies to OSS communities.

Table 16: Synthesized answer for RQ2

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Boundary</th>
<th>Study</th>
<th>Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2: How do organizational boundaries impact coordination of development activities in DSD?</td>
<td>Inhibit via collaboration policies, team organization, engineering process, and development practices</td>
<td>Vendor vs. vendor Development center vs. headquarter Vendor vs. client OSS community</td>
<td>S3, S4, S5</td>
<td>Literature review, Interview, observation</td>
</tr>
<tr>
<td></td>
<td>Mediate the impact of geographical and temporal boundaries</td>
<td>Development center vs. headquarter Vendor vs. client</td>
<td>S5</td>
<td>Interview</td>
</tr>
</tbody>
</table>
3. Multi-team large scale software project: there is a large amount of coordination needs across organizational boundaries. Different types of technical dependencies spread through organizational boundaries and require a combination of appropriate coordination mechanisms.

4.2.2. RQ2: How do organizational boundaries inhibit team coordination and project outcomes in DSD?

Thematic analysis of findings from Study S3, S4 and S5 shows two ways that organizational boundaries inhibit team coordination:

1. Four types of issues can occur in organizational boundaries: collaboration policies, team organization, engineering process, and development practices. The occurrence of collaboration policies are more likely to occur in a vendor-to-vendor relationship, while other issues can happen in all types of DSD contexts.

2. Organizational boundaries occur in the context of GSD and interact with other global boundaries to impact team coordination. An inshore outsourcing project might have more types of coordination problems than an onshore insourcing project does. Geographical and temporal boundaries also inhibit some cross-boundary coordination mechanisms.

4.2.3. RQ3: How coordination practices and tools can support coordination of technical tasks across organizational boundary in DSD?

Qualitative analysis of findings from Study S4 and S5 reveals two organization-boundary-cross coordination mechanisms:

1. Emergent and formally assigned boundary spanner. Firstly, a boundary spanner role needs to be available from several perspectives. In practice, the boundary spanner should be capable of attracting team member recognition, holding different types of expertise, decision-making ability, and work time flexibility. Boundary spanners perform four types of activities that resolve team interdependencies, which are mediating task dependency, mediating status information, leveraging global boundaries, and mediating practice flow.

2. Configuration management systems. An issue tracking system serves as a coordination mechanism by supporting code forking, notification of duplication work and a facility for boundary-cross review meetings. A version control system provides a mechanism to allow parallel development on forks to delay coordination needs. Besides, cloning MR is an effective mechanism to inform developers about duplicate work across platforms. Last but not least, formal review meetings with participants from different projects are important mechanisms for spreading project status and common issues across platforms.

The summaries of findings for RQ1, RQ2 and RQ3 are presented in Table 15, Table 16 and Table 17 correspondingly.
Table 17: Synthesized answer for RQ3

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Boundary</th>
<th>Study</th>
<th>Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ3: How can coordination practices and tools support coordination of development activities across organizational boundaries?</td>
<td>Boundary spanners mediate task dependency, status information, handle impacts of global boundaries, and navigate practice flow</td>
<td>Vendor vs. vendor Development center vs. headquarter Vendor vs. client OSS community</td>
<td>S4</td>
<td>Interview, Observation,</td>
</tr>
<tr>
<td></td>
<td>Configuration management systems temporize coordination needs resolution, support identifying and resolving duplicate dependencies and facilitate cross boundary meeting</td>
<td>Development center vs. headquarter</td>
<td>S5</td>
<td>Interview, Observation, Issue report, Code commit</td>
</tr>
</tbody>
</table>
5. Discussion and Evaluation

This chapter will discuss the results of the performed studies. Implications of the findings are both contribution to research community (Section 5.1) and recommendations to practitioners (Section 5.3). The concrete research questions will not be discussed, since these have already been covered in Section 4. Moreover, evaluation of the research will be presented via a discussion of the main threats to validity and how they were managed in our studies.

5.1. Discussion of Findings vs. Existing Literature

First of all, the thesis contributes directly to the GSD research. In particular, it is a contribution to our understanding about coordination in OSS-based development, global boundaries in GSD, adoption of boundary spanning in DSD, socio-technical coordination, and the adoption of coordination mechanisms in SE. Table 18 summaries the contributions to each research area.

Table 18: Summaries of contributions and their novelty

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Description</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1. Identification of challenges and solutions for resolving OSS component integration mismatches in commercial companies</td>
<td>Extend knowledge about OSS-based development in requirement engineering, bug fixing and collaboration practices Confirm the status of commercial contribution to OSS projects</td>
<td>OSSD</td>
</tr>
<tr>
<td>C2. Synthesized impact of geographical and temporal boundaries on team performance and software quality in DSD</td>
<td>Explain some contradict results about impact of global boundaries on project outcomes among single studies</td>
<td>Global boundaries in GSD</td>
</tr>
<tr>
<td>C3. A classification framework presenting organizational boundaries that inhibit team coordination activities during software development in DSD</td>
<td>Ground the framework from four different contexts: client-vendor relationship, vendor-vendor relationship, acquired-acquisition relationship, and companies in software ecosystems</td>
<td>Global boundaries in GSD</td>
</tr>
<tr>
<td>C4. A description of boundary spanners’ capacities and activities in coordinating cross-organization development tasks</td>
<td>Adapt theory of boundary spanning when considering global boundaries Compare and contrast emergent and assigned boundary spanners</td>
<td>Coordination in SE Boundary spanning in GSD</td>
</tr>
<tr>
<td>C5. Recommendations to management and coordination of development tasks in multiple platform systems</td>
<td>Present a novel list of development and coordination practices with regards to global boundaries in multi-platform software development context Propose a novel measure of multi-platform development to validate effectiveness of coordination mechanisms Present a list of coordination mechanisms via a configuration management system</td>
<td>Socio-technical coordination in SE</td>
</tr>
</tbody>
</table>
5.1.1. Contribution to Research on OSSD

Our major contribution to the research in OSS is C1, the description of collaboration issues and solutions for OSS component mismatches from commercial companies’ OSS adoption process:

C1. Identification of challenges and solutions for resolving OSS component integration mismatches in commercial companies

Previous work on OSS have shown insight as to how several organizations actually adopt OSS (Øyvind Hauge et al. 2008, 2009, 2010, Ayala, Cruzes et al. 2009). Our study continues this research line by investigating the practices of adopting OSS components and collaboration between commercial companies and OSS communities. We conducted a survey on OSS adoption as it has been done for COTS components in Italian software industry (Torchiano, Di Penta, et al. 2008). We explored the collaboration practices and issues between the commercial developers and OSS communities. One of the main drivers for participation in OSS projects is cost reduction through open-sourcing to an external community. It is a common perception of an ongoing shift in OSS projects as a community of individual developers, to a community of commercial organizations, often SMEs, operating as symbiotic ecosystems in a spirit of cooperation (Fitzgerald, Ågerfalk, et al. 2005, Ågerfalk, Fitzgerald, et al. 2006, Brian 2006). Our findings show that there is limited contribution from companies to the communities. The company-community contribution is rather a commensalistic relationship as the organizations mainly benefits from the community. This confirms previous findings suggesting that most organizations seem to have a rather limited contribution to the OSS communities (Bonaccorsi, Giannangeli, et al. 2006, Hauge, Ayala, et al. 2010, Chen, Li, et al. 2008).

5.1.2. Contribution to Research on Global Boundaries in GSD

Our contributions to GSD are threefold. Firstly, we present a systematic literature review that synthesize the conceptualization, operationalization, and impact of global dispersion dimensions on project outcomes:

C2. Synthesized impact of geographical and temporal boundaries on team performance and software quality in DSD.

Literature reveals that global distributed development is a multi-facet phenomenon and requires a sophisticated analysis of several dimensions (such as cultural, geographical, administrative, and economical dimensions) to better inform the feasibility of making international investments even within a new economic climate that promotes increased global interaction (Noll, Beecham, et al. 2010, Ågerfalk, Fitzgerald, et al. 2008). Although there is a large amount of GSD empirical studies, it is not clear what the impact of these global dispersion dimensions have on the project outcomes. Empirical studies show that the results contradict on how global boundaries impact coordination and project outcomes (Bird, Nagappan, et al. 2009; Ekrem, Thomas, et al. 2013, Herbsleb and Mockus 2003). For instance, Herbsleb investigated task resolution time in global software projects and found that geographical dispersion led to lower team performance (Herbsleb and Mockus 2003). However, a recent study by Kocaguneli suggested that geographical dispersion had no impact on team performance (Ekrem, Thomas et al. 2013). Our results show that the global dispersion dimensions are consistently conceptualized, but quantified in many different ways.
One of the main findings is that temporal dispersion has a positive impact on the actual team performance, but has a negative impact on the perceived performance by project managers. It implies a mismatch between the perception and the actual impact of time zones in project outcomes.

Secondly, we present a comprehensive framework of organizational boundaries across different DSD cases:

\[C3.\text{ A classification framework presenting organizational boundaries that inhibit team coordination activities during software development in DSD.}\]

An organizational boundary implies differences in identities, practices, and resources that can occur in any level of organization, such as between functional departments, development centers, and different companies. The GSD research literature has investigated different aspects of organizational boundaries, however there is no systematic and comprehensive description of these aspects. From the organizational level, appearance of different companies introduces issues of contractual obligations, goal conflicts, knowledge integration, and different work processes (Espinosa, DeLone, et al. 2006). In a vendor-client relationship, organizational boundary raises issues of selecting suitable coordination mechanisms and strategies, and measuring its impact (Gopal, Espinosa et al. 2011; Liberatore and Luo 2007; Gopal and Gosain 2010; Šmite, Wohlin et al. 2010). While previous studies mainly are based on relationships between client and vendor companies, our framework is grounded from four different contexts, client-vendor relationship, vendor-vendor relationship, acquired-acquisition relationship, and companies in software ecosystems.

Thirdly, we discussed a list of multi-platform software development issues as a foundation for a guideline of managing multi-platform software development:

\[C5. \text{ Recommendations to management and coordination of development tasks in multiple platform systems.}\]

Our study 5S explores a new context of software development – multi-platform systems. The platform boundary often crosses organizational boundaries, such as team, functional unit, and organization. There are only few existing studies that focus on this topic, however they did not explore the global boundary aspect of multi-platform software development. Joorabchi et al. explored challenges in developing mobile apps and revealed several issues about multi-platform development (Joorabchi, Mesbah et al 2013). Perry et al. conducted an observational case study of parallel development in a legacy multi-platform software system and found that current tools, processes and project management support for this level of parallelism is inadequate (Perry, Siy, et al. 1998). Our study shows that with support of collaboration tools and modern VCS, multi-platform development still faces with challenges of duplicated work across boundaries, procedural divergence, and developer-tester collaboration.

5.1.3. Contribution to Adoption of Boundary Spanning in GSD

We adapted theory of boundary spanning into GSD context to identify capacities and activities of a cross-boundary coordinator:

\[C4. \text{ A description of boundary spanners’ capacities and activities in coordinating cross-organization development tasks.}\]
Boundary spanning is a concept that has been extensively studied in different disciplines, i.e., Organization Science, Social science and IS. Boundary spanner’s capacities are found to be positively correlated with team performance as well as a higher rating of group cohesion (Ancona and Caldwell 1992). Boundary spanners, either in the form of individuals or teams, can significantly improve the collaborative process within global project (Di Marco, Taylor et al. 2010). In our study, we identify the common capacities and necessary activities that a boundary spanner possesses. Our category differs from existing frameworks of boundary spanning activities (Tushman and Scanlan 1981a; Brown and Duguid 1998), that it is specialized in GSD with considering the occurrence of different dimensions of global context. We also compare and contrast the role of emerged boundary spanners and the assigned ones (Levina and Vaast 2005).

5.1.4. Contribution to Socio-technical Coordination Research in SE

Firstly, we validate the common input variables on socio-technical coordination to predict project outcomes by social network analysis of issue resolution time using proxies of organizational boundaries and developer’s feature. Large amount of SE studies empirically evaluate measurements of communicated information, such as number of comments on a bug, number of message in mailing list, number of related authors in a bug report, number of developer’s commit by their relationships with communication, and bug fixing time (Pinzger, Nagappan et al. 2008; Bettenburg, Schrotter, et al. 2008; Abreu and Premraj 2009; Bird, Nagappan et al. 2009; Schrotter, Wolf et al. 2010; Anbalagan and Vouk 2009; Feczak and Hossain 2009; Guo, Zimmermann, et al. 2010). We confirm the findings from these studies and also introduce a new metric based on average past resolution time of a developer. The relationship between average past resolution time and current issue resolution time, together with the inverse relationship between developer’s experience and issue lead time, might be evidence of a learning process. For a specific developer, the average time taken to resolve an issue is shortened when gaining experience in resolving issues. However, this trend varies between developers and projects, as indicated by the variety of correlation coefficient values among different projects.

Secondly, we contribute to the field by describing the new type of dependencies and how they are resolved using configuration management systems:

*C5. Recommendations to management and coordination of development tasks in multiple platform systems.*

Software developers have different types of coordination needs; coordination across sites is more challenging than within a site (Espinosa, DeLone, et al. 2006). Different types of dependencies among developers and software artefacts might lead to different coordination needs. Our studies reveal a new type of technical dependency across different software platforms: divergence of identical code files. While early identification of dependencies among files and developers is emphasized, finding a duplicated link among these technical dependencies are important to reduce redundant development effort.

Thirdly, our work also contributes to a discussion of various types of coordination mechanisms in a GSD context. Cataldo et al. emphasized the importance of early identification of the “right” set of technical dependencies that drive coordination requirements among software developers (Cataldo, Hersleb, et al. 2008). However, there are not always
attempts to solve coordination requirements at the beginning. We revealed that coordination requirements are temporized in complex inter-dependencies and under the constraints of the project schedule.

5.1.5. Contribution to Coordination Mechanisms in SE

Paasivaara et al. identified successful collaboration practices in inter-organizational projects such as: milestone synchronization, frequent deliveries, and the establishment of peer-to-peer links. Some other coordination mechanisms that might be useful for collaboration across organizational boundaries are given in Table 2, Section 2 (Paasivaara and Lassenius 2003). Our findings complement the effective coordination practices by the introduction and evaluation of two mechanisms: the boundary spanner role and cloning mechanism in configuration management systems:

- **C4. A description of boundary spanners’ capacities and activities in coordinating cross-organization development tasks.**

Previous studies highlight the importance of mutual adjustment when strong coordination needs occur (Wageman 1995; Mirani 2007; Šmite, Moe, et al. 2008), as more complicated inter-dependencies and complexity requires more informal communication. Our studies in different GSD contexts agree with these findings, by highlighting the role of a boundary spanner, either informal or formal roles as an important inter-organizational coordination mechanism.

Kortlarsky et al. investigated different types of coordination mechanisms, suggesting that technology-based mechanisms are most useful for amplifying knowledge management processes to allow knowledge sharing (Kortlarsky, van Fenema, et al. 2008). Kiesler and Cummings found that technology is important, but insufficient to alone recreate the same facilitating environment in distributed teams that is present in co-located settings (Cummings and Kiesler 2005). Lar Bendix et al. suggest that many GSD challenges could be alleviated by configuration management system techniques (Bendix, Magnusson, et al. 2012). Our studies confirm the usage of configuration management systems as an effective coordination mechanism.

Coordination is more difficult because of the complex and redundant inter-dependencies among the code artefacts and developers. The different development approaches hindered in-depth discussions on technical tasks at the team level. Development infrastructures, such as development tools and test environments are also different across geographical distance and introduce difficulties for inter-team collaboration. Besides, each project has its own priority of features and bug fixes. The likely complexity of the divergent code base leads to that few engineers, if any, really understand in detail the entire code base, thus leads to lack of “who does what” and “who knows what” knowledge across the codebases.

Decisions on which type of coordination mechanisms to use should depend on the current dispersion context setting, the current team coordination technology and practices, and prioritized type of interdependencies. Although managers cannot entirely surmount the challenges associated with bridging these boundaries, they can mitigate them by combining different coordination mechanisms. Problems with geographical dispersion and small temporal dispersion could be mitigated by selective, structured communication and
collaborative development tools (Lanubile, Ebert, et al. 2010). Challenges of significant temporal dispersion could be reduced by adoption of scheduling and sharing work processes across boundaries. Problems with cultural dispersion could be reduced by formal communication and early standardization activities, such as team training and site visits. The same coordination mechanism may be applicable at the individual level, but may not be suitable at a team level. Our summary shows that communication and shared artifacts should be used together as needed, and a defined process should be adopted at the team and organizational levels.

5.2. Evaluation of Validity Threats

Empirical research needs to be conducted in a credible and believable manner. Yin recognizes that validity may be increased through triangulation by using different data sources, different researchers, different theories, and different research methods (Yin 2003). The research reported here is conducted in collaboration with other researchers. It has been collected data from several different settings as an ongoing activity for the last four years. While the research has had a focus on case study research, evidence also comes from surveys, action research, and literature review. Nevertheless, this work is not without limitations. Wohlin et al. defined four categories of validity threats (Wohlin 2000), namely internal, external, construct, and conclusion validity. Table 19 presents the list of validity threats that occurs in our studies and the approaches to resolve or reduce their impact.

5.2.1. Internal Validity

Internal validity concerns the degree to which the results can be trusted, i.e., whether an investigated causal relationship between treatment and outcome is not a result of the factors measured. There are some internal threats, such as bias in selection of study subjects, data collection and analysis, and ambiguity about the direction of causal influence. Many of these threats were resolved or reduced by adopting methodological triangulation and detailed documented steps of data collection, processing, and analysis.

Bias in project/team selection: occurs during case study research. Engagement of industrial partners in research is a typical challenge for the SE community. Our population is DSD projects with occurrence of a type of organizational boundary and with coordination challenges. For OSS projects, a preliminary analysis of suggested projects were performed to ensure they were relevant. Decision about the project’s relevancy was done by reading through project descriptions, issue comments, and contacting some contributors. For commercial projects, some were selected by talking to their project managers, and some were found by suggestion from colleagues and supervisors.
Table 19: Threats to validity and solution

<table>
<thead>
<tr>
<th>Validity threats</th>
<th>Category</th>
<th>Study</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias can occur in project/team selection</td>
<td>Internal validity</td>
<td>S1, S2, S4, S5</td>
<td>Select wide range of DSD projects Preliminarily analyze candidate projects</td>
</tr>
<tr>
<td>Bias in data collection</td>
<td>Internal validity</td>
<td>S1, S4, S5</td>
<td>Thorough plan and review study designs by external experts</td>
</tr>
<tr>
<td>Misinterpretation during interviews</td>
<td>Internal validity</td>
<td>S1, S4, S5</td>
<td>Send interview questions beforehand Include more than one interviewers Send post-interview clarifying questions</td>
</tr>
<tr>
<td>Identification of commercial company representatives</td>
<td>Internal validity</td>
<td>S1</td>
<td>Manually check multiple sources to determine developer’s affiliation</td>
</tr>
<tr>
<td>Compounding factor in quantitative model</td>
<td>Internal validity</td>
<td>S2, S5</td>
<td>Include context factors into quantitative models</td>
</tr>
<tr>
<td>Missing repository data</td>
<td>Internal validity</td>
<td>S2, S5</td>
<td>Delete invalid data points with considering the significance of analysis bias</td>
</tr>
<tr>
<td>Generalization to software development</td>
<td>External validity</td>
<td>S1, S2, S3, S4, S5</td>
<td>Determine target population when design study Consider all context factors when drawing conclusion about generalization</td>
</tr>
<tr>
<td>Credibility of findings</td>
<td>External validity</td>
<td>S5</td>
<td>Iteratively provide and receive feedback from relevant stakeholders</td>
</tr>
<tr>
<td>Operationalization of concepts</td>
<td>Construct validity</td>
<td>S2, S5</td>
<td>Use literature and initial interviews to provide reasoning for constructing measures</td>
</tr>
<tr>
<td>Causal-effect relationship</td>
<td>Conclusion validity</td>
<td>S2, S5</td>
<td>No attempt to draw a causal relationship</td>
</tr>
</tbody>
</table>

Bias in qualitative data collection: occurs during the selection of interview or survey participants, by selecting irrelevant candidates. This threat was solved by a thorough and peer-review study design. For selecting survey participants (S1), the invitation letter and questionnaire were sent to the candidate participants for confirmation of relevance. For selecting relevant interviewee (S4 and S5), we talked to people at the management level and to project experts to have appropriate references. In S5, selection of new interviewees was complete when no more new information was added from the interviews.

Misinterpretation during interviews: to ensure that the presented concepts are understood correctly by the interviewees, we sent the interview guide with a detailed description of the survey to the interviewee beforehand (S1 and S4). The interview guide was carefully reviewed by my colleagues to ensure consistent understanding among us (S1 and S5). During most or all interview sections, there were at least one more interviewer in addition to the author (S1, S4 and S5). During the interviews, we tried to ensure the interviewee understood what they were asked. The taped audio of interview was randomly checked by one of my colleagues to ensure the data quality. After the interview, any confusion would be clarified with interviewee.
Identification of commercial company representatives: in OSS projects, email and alias of a contributor might not accurately reflect the affiliation of the contributor (study S2). The classification of contributor type was manually executed based on the (1) contributor list, (2) mailing list, (3) the contributor’s online profile, and (4) the stakeholder’s email with a private company domain. We also did some confirmation by directly contacting suspect contributors. Although a major amount of the stakeholder’s affiliations were identified, there were still some stakeholders with no company information. However, this group of unidentified stakeholders is responsible for a very small portion of issues in general. Since the major portion of the issues comes from identified stakeholders, the comparison of resolution time between groups of stakeholders would not be significantly influenced.

Compound factor in quantitative model: is a threat in quantitative studies, as it is hard to distinguish the effect of two factors from each other. Different from experiments, a case study has little control over context factors that might impact the desirable observation. Therefore, it is a typical problem of quantitative models based on repository data from actual projects. One way to reduce this threat is to include potential confounding factors that may affect the relationship into a quantitative model. For instance, S2 included measures of the past performance of the developer and MR characteristics in addition to the measure of developer communication. In S5, project duration, project size, and several other predictors were included, which affect the investigated relationship.

Missing quantitative data: the collection of quantitative data in SE often involves substantial effort and often leads to missing or invalid data (Audris 2008). There is a concern with the quality of data collected from issue tracking systems or version control systems, such as missing fields in bug reports, duplicated files, and inconsistent person identification. This threat was minimized by a careful selection of datasets in each study. We also removed bad and incomplete data points. The deletion of invalid data points was checked so it would not create significant bias in data analysis (S2 and S5). In S5, we removed some old projects due to missing values and narrowed conclusions to explicitly exclude them. Complicated approaches to deal with missing value, such as adding observations that may explain why the values are missing, substitution techniques to fill missing values were not considered in our studies.

5.2.2. External Validity

External validity concerns the degree to which the results can be generalized to an overall population and setting. The primary threat to external validity is that the study is based on a few and possibly not typical projects. Challenges with external validity can come from product/people/project (the subjects are not representative for the population), place (the project setting is not representative), and time (the study is carried on in a special period of time). In general, most empirical studies in the industry suffer from non-representative participation. This threat is discussed for the different RQs, as they used different sets of projects.

RQ1, which discovers coordination practices and issues in DSD, is (partly) answered by all of the included studies. Consequently, collaboration challenges, issues, and practices are collected from various types of distributed context settings, such as OSS ecosystems, acquired and acquisition companies, and client and outsource companies. The projects also vary in size, application domain, development life cycle, and team configuration. One commonality
among these projects is adoption of a collaborative infrastructure, such as SVN, Jira and mailing lists. Therefore, issues with coordination of technical tasks are generalized to a heterogeneous set of DSD projects that use modern configuration management systems.

RQ2, which explores perspectives of organizational boundaries, is addressed by S3, S4 and S5. Results were collected from four case studies involving commercial companies in Norway, Japan, Vietnam, and USA, and one case study of two OSS projects. The projects were selected to represent different global arrangement settings, such as insourcing, offshore outsourcing, and inshore outsourcing. OSS projects are considered as an extreme case since it is inherently different from the other three project types in term of the number of companies in the collaboration relationship, economic and strategic motivations and constraints. Therefore, the occurrence of these organizational boundaries can be generalized to a heterogeneous set of DSD projects that use modern configuration management systems.

RQ3, which investigate effectiveness of specific coordination mechanisms, is addressed by S4 and S5. Boundary spanners, as a coordination mechanism, were found not only in commercial projects, but also in OSS projects. Common abilities and activities were identified across four case studies in S4 and repeated in S5. Cloned MR is a typical mechanism in Avaya, hence it is not possible to generalize its effectiveness to other types of organizations.

In S5, we performed action research. A main threat to validity of this research is to convince industrial partners who did not participate in this research, that the results are believable. With action research, the coordination mechanisms and practices were influenced by the researcher by introducing these to practitioners. The study was conducted as part of an on-going internal research project; hence it is straightforward to take into account the research findings as room for practices and processes improvement. Parts of the findings, such as the description of extent of parallel development, were presented to customers and received positive feedback.

5.2.3. Construct Validity

Construct validity concerns the relationship between theory and observation. This threat is concerned with quantitative studies with testing of relationships between theoretical concepts. Possible threats are, e.g., case selection bias (a single case may not reflect the constructs), operationalization bias (a single type of measure may be misleading), and hypothesis guessing. Construct validity is also concerned with using the right tools and metrics for gathering the data. In our studies, the investigated and measured constructs (i.e., communication frequency, coordination mechanisms, defect proneness, and organizational boundaries) are based on well-founded concepts in SE, IS, and Organization Science. Hypotheses were proposed based on (1) reasons extracted from a thorough literature review, (2) discussion with project members to check the meaningfulness of the hypotheses, and (3) brainstorm sections with co-authors. Multiple alternative reasons and explanations for a hypothesis were also taken into account.

In S2, communication among developers was operationalized by measures of number of comments, messages, and number of issue-involved stakeholders. Although collaboration between stakeholders can be done via other channels such as IRC, Skype, and face-to-face discussion, issue tracking systems and mailing lists are the most common discussion means. The most relevant discussion about an issue should be found in these channels. Besides, these measures are commonly used in SE research on team collaboration and found to be useful to
predict the outcomes of several projects (Herbsleb and Mockus 2003, Cataldo, Hersleb et al. 2007, Cataldo, Bass et al. 2008).

In S5, the concept of coordination across platforms is operationalized by measures of amount of MRs cloned, and the number of boundary spanners in the project. Interviews with project staff confirmed the popular adoption of these approaches across platforms. We were not able to collect data about the third coordination approach, MR review meetings, for all relevant projects. There might be other ways to coordinate but they do not focus on the cross-platform perspective. However, we believe that the proposed measures are sufficient to describe the overview of team coordination across platforms. It can be argued that a complete version of these measures, by weighting other factors, such as MR length and performance of the boundary spanners, might be more accurate proxies of the concept. However, the basic version of the measures reduces the compound effect of other coordination aspects and provides a simple view of the concept.

5.2.4. Conclusion Validity

**Conclusion validity** concerns whether or not the right conclusions are drawn based on the collected data. The conclusion validity is in experiments and qualitative analyses concerned with ensuring (significant) statistical relationships between the treatment and outcome, or the independent and the dependent variables. One threat to this validity is the claim of statistical causal relationship, i.e., between the extent of collaboration and issue resolution time, or software quality.

In S2, we did not claim a causal relationship between measures of extent of collaboration and bug resolution time, as we performed correlation analysis rather than conducting impact studies. For example, even though we observed a relationship between the number of stakeholders and issue resolution time, the larger number of involved stakeholders could be caused by the size and complexity of the issue, which would also cause the resolution time of the issue to be longer. Besides, in the multivariate model, we assumed that the relationship among variables takes a linear form; however, the actual relationship can be better represented by another mathematical formula, such as non-linear regression models.

5.3. General Recommendations for Practitioners

Twelve recommendations are extracted from our five studies, as shown in Table 20:

- **R1.** Using measures of developer’s past performance to predict issue resolution time: developer’s past performance on issue resolution is a helpful indicator of future performance. Besides well-investigated measures of communication and developers’ social network positions, the past performance should be considered when building a model of socio-technical coordination. A discussion of the measures in comparison with existing measure based on historical data is given in Paper MP1 and SP1.

- **R2.** Involving external stakeholders in resolving OSS component mismatches: it is common that a selected OSS component does not provide a fully match of functional and non-functional requirements. While a commercial company can develop the component within the organization boundary, the company, the local code commits and bug fixes can lead to conflicts with the component mainstream enhancement, which might not be resolved. Our study suggests that the local version of the adopted
OSS component should be kept synchronized with the mainstream development. Besides, the participation of customers tends to improve functional mismatch resolution, while the feedbacks from OSS communities tend to improve non-functional mismatch resolution. Further discussion on this suggestion is given in Paper MP2 and MP3.

**R3.** Alignment of perception on distributed work across time-zones with its actual impact: empirical evidence reveals that in many projects, managers tend to have a pessimistic view on being distributed over physical space and time zones. Measurement based on historical data might provide a precise reflection on the impact of temporal boundaries on team performance. Besides, given the significant role of coordination as a project success factor, managers should invest on team configuration in early phases of the project as an establishment for team coordination in later phases. The recommendation comes from a synthesis of existing primary studies, which is presented in Paper MP2 and SP2.

**R4.** Considering the influence of global dispersion at different organizational levels: at the individual and team level, lack of face-to-face interaction due to time-zone difference, directly and negatively affect team performance. At the project and company level, these negative impacts might be neglected due to other prioritized organizational goals and policies (Paper MP4). This issue should be taken into account when aligning the objective of individual developers and teams with organizational goals. The further discussion of this recommendation is described in Paper MP4 and SP2.

**R5.** Collocating development teams for quality-critical software components. Our studies revealed a negative impact of geographical and temporal dispersion on software quality reported from primary studies (Paper MP4). Therefore, distributing teams across global boundaries might not be a good development approach for quality-critical software products. Traditional quality assurance approaches in collocated teams might need to consider the differences in time zone, lack of face-to-face communication, and a variety in processes and practices among locations. The recommendation is based on a synthesis of existing primary studies, which is presented in Paper MP4 and SP2.

**R6.** Guiding organic coordination activities by enforcing mechanistic coordination: Mechanistic coordination is as important as organic coordination in mediating and improving project performance (Paper MP4). Consequently, the management effort should not only put into creating of informal communication communities among projects, but also establishing a utilized formal coordination mechanisms across these projects. Relying on a single coordination mechanism is not sufficient to support team collaboration beyond the organizational boundary. The formal processes, standards, and guidelines should be established among the organizations to facilitate the transparent collaboration activities at both management and operation level (Paper MP6). This suggestion has been observed in existing studies and discussed in Paper MP4 and MP6.

**R7.** Considering both technical requirements of coordination and the organizational policy of competition. There is often a tradeoff of satisfying both technical coordination needs and organization’s benefits in inter-organizational projects (Paper MP5 and MP6). Team collaboration, which initiates from a technical coordination requirement, is influenced by technical, social, and organizational factors (Paper SP3). Meanwhile,
competition attitude, which comes from business strategy and organization’s benefits, might inhibit technical collaboration. Therefore, collaboration attitudes should be considered when establishing the coordination infrastructure, process, and practices. The discussion of this recommendation is given in Paper MP6 and SP3.

R8. Recognizing the coordination role of in-practice boundary spanners: boundary spanners are present in every software projects in various forms (Paper MP6). However, boundary spanners are not always recognized for their coordination activities, especially in case of emergent boundary spanners. While boundary spanners are an important team coordination mechanism, they also face some issues, such as knowledge silos, role conflicts, and stress. Therefore, the strengths and weaknesses of adopting boundary-spanning activities should be considered to allocate sufficient supports. The goals of the boundary spanners should be recognized and aligned with goals of the organization. The role of boundary spanners and their coordination activities are discussed in the Papers MP6, MP7, and MP8.

R9. Automating the identification of redundant work across platforms: our studies show that there is a significant amount of files under parallel development in multiple platform systems (Paper MP8). Moreover, there is a lot of related files that are changed at the same time without awareness to files’ owners. A tool that identify all related files under parallel development for specific projects, will help developers to reduce the possible redundant works and prevent integration conflicts in later phases. A description of a desirable tool, based on developer’s requirement, is given in MP8.

R10. Considering organizational-technical issues when creating forks across related projects: on one hand, forking is an unavoidable action in DSD. The issue is whether to maintain forks according to the architectural structure or the organizational structure (Shihab et al. 2012). One strategy to create a diversified codebase is to dedicate one fork per component. Doing so, allow software components to be developed in isolation. However, in certain cases, multiple components are modified in a single fork, causing forks to cross-cut the architecture. Another strategy is to create separate forks for each organizational team. This approach allows better teamwork, code ownership, and management of the local teams. The decision on what to fork and when to fork depends on the project context, such as schedule, technical variation, required coordination and organizational policy. S5 revealed that a codebase is often forked based on architectural modularity in an intra-project context, and it is often forked by the organizational and managerial structure. A discussion on existing branching and forking strategies is given in Paper MP7 and MP8.

R11. Establishing a united code merging process across interdependent projects: merging code is a complicated activity in DSD projects, because merging code among development branches might involve multiple autonomous development teams. To resolve this problem, there should be a united process and policy on how, and in which order, code should be merged from branches. The discussion of this recommendation is given in Paper MP8.

R12. Inform about changes of duplicate work across functional units, such as between testing team and development team. Our studies show that high-level testing teams are often not sufficiently informed of changes in common features across different platforms. Especially duplicate work, i.e., bug fixes or repeated functionality are not well-informed to testing teams. Duplicate test cases can be reduced by better
communication of these changes. An example is to deploy the test environment together with the main codebase, as a bug can occur in different operating contexts (application vs. factory test), and in different phases of operation: boot, initialization, and upgrade. The necessary information to reproduce the bug should be transparent between both teams. The discussion of this suggestion is given in Paper MP8.

Table 20: Summary of recommendations

<table>
<thead>
<tr>
<th>ID</th>
<th>Recommendation</th>
<th>Contr.</th>
<th>Audience</th>
<th>From</th>
<th>Supported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Using measures of developer’s past performance to predict issue resolution time</td>
<td>-</td>
<td>Model builder</td>
<td>S2</td>
<td>Quantitative model</td>
</tr>
<tr>
<td>R2</td>
<td>Involving external stakeholders in resolving OSS component mismatches</td>
<td>C1</td>
<td>Project manager</td>
<td>S1</td>
<td>Interviews</td>
</tr>
<tr>
<td>R3</td>
<td>Alignment of perception on distributed work across time-zones with its actual impact</td>
<td>C2</td>
<td>Project manager</td>
<td>S3</td>
<td>Literature reviews</td>
</tr>
<tr>
<td>R4</td>
<td>Considering the influence of global dispersion at different organizational levels</td>
<td>C2</td>
<td>Project manager</td>
<td>S3</td>
<td>Literature reviews</td>
</tr>
<tr>
<td>R5</td>
<td>Collocating development teams for quality-critical software components</td>
<td>C2</td>
<td>Project manager</td>
<td>S3</td>
<td>Literature reviews</td>
</tr>
<tr>
<td>R6</td>
<td>Guiding organic coordination activities by enforcing mechanistic coordination</td>
<td>C3</td>
<td>Project manager</td>
<td>S4</td>
<td>Literature reviews, Interviews, Observation</td>
</tr>
<tr>
<td>R7</td>
<td>Considering both technical requirements of coordination and the organizational policy of competition</td>
<td>C3</td>
<td>Project manager</td>
<td>S4</td>
<td>Literature reviews, Interviews</td>
</tr>
<tr>
<td>R8</td>
<td>Recognizing the coordination role of in-practice boundary spanners</td>
<td>C4</td>
<td>Project manager</td>
<td>S4, S5</td>
<td>Literature reviews, Interviews, Observation</td>
</tr>
<tr>
<td>R9</td>
<td>Automating the identification of redundant work across platforms</td>
<td>C5</td>
<td>Software developer</td>
<td>S5</td>
<td>Interviews, Quantitative descriptive</td>
</tr>
<tr>
<td>R10</td>
<td>Considering organizational-technical issues when creating forks across related projects</td>
<td>C5</td>
<td>Software developer</td>
<td>S5</td>
<td>Interview</td>
</tr>
<tr>
<td>R11</td>
<td>Establishing a united code merging process across related projects</td>
<td>C3</td>
<td>Project manager</td>
<td>S4, S5</td>
<td>Literature reviews, Interviews</td>
</tr>
<tr>
<td>R12</td>
<td>Informing changes of duplicate work across functional units</td>
<td>C5</td>
<td>Project manager, Software developer</td>
<td>S3, S4, S5</td>
<td>Literature review, Interview</td>
</tr>
</tbody>
</table>
6. Conclusions and Future work

This thesis has presented the results from several empirical studies executed during four years of Ph.D work. The studies combine literature study, collecting and analyzing quantitative data from configuration management systems, and qualitative data from various sources, qualitative surveys, case studies, and action research. A mixed-method research design was applied to allow taking benefit of all available data, combining the results and answering questions that are not possible to answer otherwise. The previous chapter discussed the answers to the three research questions posed in Chapter 1. This final chapter presents the conclusions regarding the overall research problem and questions, and lists the contributions made by this thesis.

6.1. Summary of Findings

Coordination is an important research topic in SE and particular DSD. When projects spread through organizational boundaries, such as projects involving multiple companies and autonomous development units, it appears different types of coordination problems which requires suitable coordination mechanisms. Motivated by the importance of coordination in software development, the fundamental differences of coordination activities in context of an inter-organizational and intra-organizational software projects, the overall problem put forward in this thesis was:

How can coordination of software development activities across organizational boundaries be supported?

We investigated collaboration activities in general in a DSD context. It appears that organizational boundaries might be meaningful in different project contexts and for different types of software development processes and activities. For instance, it seems that organizational affiliation does not impact the issue resolution time in OSS ecosystems. The direction of collaboration is also unbalanced in OSS-based commercial software projects, because there is a limited contribution from commercial companies back to OSS community. However, organizational boundaries do matter in some other cases, such as vendor-to-vendor relationships.

We explored common challenges with coordination activities in variety of DSD software projects, and identified four aspects of organizational boundaries, which are organizational challenges, collaboration policies, team organization, engineering process, and development practices. The severity of each aspect varies and is related to the amount and type of dependencies available across the boundaries. For instance, organizational boundaries might leads to issues of team awareness, trust, responsibility, and redundant work in some cases. Moreover, organizational boundaries can combine with other global boundaries, i.e., temporal and geographical boundaries to limit the availability of some coordination mechanisms.

We identified two effective boundary-cross coordination mechanisms in the DSD context and described how they can be adopted in a specific DSD context. The role of boundary spanners, as a default mechanism in every project, needs to be explicitly recognized as a coordination mechanism. Both emergent and assigned boundary spanners support the mediation of task
dependencies, navigation of status information, management of global boundaries, and facilitation of practice exchanges. Configuration management systems have a large potential to solve many DSD coordination problems. An issue tracking system facilitates the notification of duplication work and boundary-cross review meetings. A version control system allows parallel forks to delay coordination needs.

Our findings have a direct implication for practitioners. For project managers, developers and researcher, we offered twelve recommendations for improving coordination across organizational boundaries:

**R1.** Using measures of developer’s past performance to predict issue resolution time  
**R2.** Involving external stakeholders in resolving OSS component mismatches  
**R3.** Alignment of perception on distributed work across time-zones with its actual impact  
**R4.** Considering the influence of global dispersion at different organizational levels  
**R5.** Collocating development teams for quality-critical software components  
**R6.** Guiding organic coordination activities by enforcing mechanistic coordination  
**R7.** Considering both technical requirements of coordination and the organizational policy of competition  
**R8.** Recognizing the coordination role of in-practice boundary spanners  
**R9.** Automating the identification of redundant work across platforms  
**R10.** Considering organizational-technical issues when creating forks across related projects  
**R11.** Establishing a united code merging process across interdependent projects  
**R12.** Informing changes of duplicate work across functional units

**Table 21: Suggestions for future work**

<table>
<thead>
<tr>
<th>Future work direction</th>
<th>Possible study type</th>
<th>Area</th>
<th>From study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supporting decision-making process of OSS component selection</td>
<td>Quantitative survey</td>
<td>OSSD</td>
<td>S1</td>
</tr>
<tr>
<td>Adapting OSS project coordination practices on commercial GSD software projects.</td>
<td>Comparative case studies</td>
<td>GSD</td>
<td>S1, S2, S3, S4</td>
</tr>
<tr>
<td>Combined effect of global dispersion dimensions on project outcomes</td>
<td>Confirmative case studies</td>
<td>GSD</td>
<td>S3</td>
</tr>
<tr>
<td>Quantifying organizational boundary and its impact on project outcomes</td>
<td>Confirmative case studies</td>
<td>GSD</td>
<td>S4</td>
</tr>
<tr>
<td>Coexistence of collaboration and competition in GSD projects</td>
<td>Exploratory case study Ethnography</td>
<td>GSD</td>
<td>S4</td>
</tr>
<tr>
<td>Evolution of matching coordination needs and coordination mechanism over time</td>
<td>Exploratory case study Longitudinal case study</td>
<td>GSD</td>
<td>S4, S5</td>
</tr>
<tr>
<td>Context based adoption of boundary spanners</td>
<td>Exploratory case study</td>
<td>GSD</td>
<td>S4</td>
</tr>
<tr>
<td>Investigation of codebases divergence and merging practices.</td>
<td>Exploratory case study Longitudinal case study</td>
<td>Multiple platform SE</td>
<td>S5</td>
</tr>
</tbody>
</table>
6.2. Future Work

Team coordination in DSD will continue to be an important research area. We provide eight suggestions for future works in research on coordination in DSD, as shown in Table 21:

1. Supporting a decision making process in OSS component selection. The main purpose of the process is to find out indicators of components reliability and maintainability from the OSS component community. Besides, some of the context factors show potential impact on requirement mismatch resolution decisions, such as source of requirements or reliability of components. However, we do not have enough data to conduct a quantitative analysis on these factors. In future studies with more data points, a more quantitative analysis of impacting factors could be implemented. Last, but not least, we highlight the importance of stakeholder involvement in the mismatch resolving process. A deeper understanding of stakeholder involvement would help to improve the matching process.

2. Adapting OSS project coordination practices on commercial GSD software projects. Many OSS projects are a successful example of team coordination across heterogeneous groups of participants. Future research might investigate the extractable coordination mechanisms in OSS that can be applied in a GSD context. For instance, future research should explore more on team performance in considering dispersion dimensions and the role of OSS communication infrastructure on this relationship.

3. Combined effect of global dispersion dimensions on project outcomes. Our SLR revealed that there is a small amount of studies that consider all five dispersion dimensions. Future studies in GSD might explore the combined effect among dispersion dimensions.

4. Quantifying organizational boundary and its impact on project outcomes. Our SLR disclosed a significant amount of studies on geographical, temporal, and cultural dimensions. This trend suggests an opportunity for future research on relationship between organizational differences and work process dispersion on team performance. Particularly, this empirical evidence is in a qualitative manner. The adoption of quantitative approach on exploring the impact of organizational boundaries on project outcomes is a promising future research direction.

5. Coexistence of collaboration and competition in GSD projects. Our SLR revealed a significant amount of GSD research focusing on the 3Cs: collaboration, communication and coordination. However, another C: competition, is also important for a successful GSD project, which is not much explored. The concept of competition and how it co-exists with collaboration in a global market context, is a research area in business, marketing and engineering management. There is a term “copetition” to describe this phenomenon. However, in the SE context, our case study is the first study exploring this phenomenon. Future research can investigate the existence of competition policies in a software ecosystem where there is a collaboration among many vendors.

6. Evolution of matching coordination needs and coordination mechanisms over time. We revealed that there are few longitudinal empirical studies on the topic of coordination in GSD. Consequently, changes in coordination needs in different project phases are difficult to observe. Studies about which coordination mechanisms are adopted to solve which coordination needs that evolve overtime, are beneficial for the GSD research area.
7. **Context-based adoption of boundary spanners.** Future studies should questions about “when do boundary spanners leverage boundaries” or “when are boundary objects effectively used?”, and “what factors drive the decision-making procedure of boundary spanners when mediating task and information flow?”

8. **Investigation of codebase divergence and merging practices.** In S5, we only investigated the extent of forking activities and redundant work via commits and MRs. In future work, we would like to address code divergence and assess the number of identical files between them in a more detailed manner. In particular, we would like to investigate if there is any difference between a file that diverges much compared to a file that does not in term of software quality and coordination efforts.
References


83


Guo, P.J., Zimmermann, T., Nagappan, N., & Murphy, B. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In Proceedings of 32nd ACM/IEEE International Conference on Software Engineering (ICSE), Cape Town, South Africa


Appendix A: Full Contents of Included Papers
Paper MP1 - Impact of Stakeholder Type and Collaboration on Issue Resolution Time in OSS Projects

Impact of stakeholder type and collaboration on issue resolution time in OSS Projects

Nguyen Duc Anh¹, Daniela S. Cruzes¹, Claudia Ayala², Reidar Conradi¹

¹Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway
{anhn, dcruzes, conradi@idi.ntnu.no}

²Technical University of Catalunya, Department of Service Engineering and Information Systems, Barcelona, Spain {cayala@essi.upc.edu}

Abstract. Initialized by a collective contribution of volunteer developers, Open source software (OSS) attracts an increasing involvement of commercial firms. Many OSS projects are composed of a mix group of firm-paid and volunteer developers, with different motivations, collaboration practices and working styles. As OSS development consists of collaborative works in nature, it is important to know whether these differences have an impact on collaboration between difference types of stakeholders, which lead to an influence in the project outcomes. In this paper, we empirically investigate the firm-paid participation in resolving OSS evolution issues, the stakeholder collaboration and its impact on OSS issue resolution time. The results suggest that though a firm-paid assigned developer resolves much more issues than a volunteer developer does, there is no difference in issue resolution time between them. Besides, the more important factor that influences the issue resolution time comes from the collaboration among stakeholders rather than from individual characteristics.

1 Introduction

Open source software (OSS) development is a highly distributed and collaborative activity. In OSS projects, stakeholders, who are people involve in software development project such as developers, project leader, tester and endusers, collaborate with each other in various ways to accomplish development tasks. Although OSS was born as a movement mainly based on contributions of volunteer stakeholders, an increasing number of firms are getting involved in OSS projects [21][31]. Lakhani et al. found that around 40% of programmers are paid by companies to contribute to OSS projects [24]. Hars and Ou obtained similar results in a survey on the developers of the Linux kernel [29]. Consequently, many open source projects contain both types of stakeholder (firm-paid and volunteer), which have different motivations, collaboration practices and working styles. For instance, firm-paid developers contribute to the OSS community as part of their jobs, which provide them a financial motivation. In addition, they often also work on proprietary software since it constitutes a part of the business model of their sponsor firm [2][9][25]. Therefore, they have to learn the community working style and adjust to the rhythms and the demands of OSS development [2]. In contrast, volunteer developers are usually motivated by social or technical reasons to demonstrate or improve their technical skills [9][25].

Several studies have investigated the potential differences among firm-paid and volunteer developers in OSS projects [2][21][24][29][31]. However, these studies did not address whether these differences actually have an impact on the OSS project outcomes such as quality of the source code, productivity of developers, activeness of the community and time to accomplish a software evolution task.

A software evolution task (or software issue) is normally referred as a unit of work to
accomplish an improvement in the system. Dealing with a software issue includes fixing defects, implementing new feature requests and enhancing current system features. With a large amount of issues that occur from time to time, resolving them in a cost-effective manner is essential to achieve a high user satisfaction with less working effort.

Besides the impact of some special characteristics of stakeholders (in the issue resolving process, they are usually reporters and assignees), the issue resolution time can be influenced by a collaborative working process between reporters and assignees. Pinzger et al. mention the Coordination theory in OSS, which state that the interaction among stakeholders can impact software quality (such as mean time between failure) and work performance (such as defect removal effectiveness and problem fixing time) [30]. In the issue resolving process, stakeholders often use electronic media such as mailing list, IRC and issue tracking system to discuss, comment and clarify about an assigned task [23][26]. The collaboration among stakeholders, such as discussion, instruction and clarification on an issue, is important to the completion of the issue-resolving task.

This study has three main objectives. First, we characterize the difference in the average amount of resolved issues and issue resolution time between a volunteer assignee and a firm-paid assignee. To best of our knowledge, there is no study that empirically investigates the influence of volunteers versus firm-paid developers on issue resolution time. Second, we investigate collaboration among stakeholders in OSS projects by using Social network metrics and analysis. Last, we explore the impact of the collaboration measures on issue resolution time. While there are several studies using Social network metrics investigating software quality (as described in Section 2.1), this is among the first attempts to apply these metrics on studying issue resolution time.

The rest of the paper is organized as follows. Section 2 presents a construction of stakeholder collaboration measure using Social network analysis (SNA). While Section 3 states our hypotheses, Section 4 describes our case study and data collection procedure. Section 5 provides the hypotheses testing results. Section 6 discusses the findings and Section 7 identifies the threats to validity. The paper ends with a conclusion and future works.

2 Stakeholder collaboration measure by social network analysis (SNA)

2.1 Impact of collaboration on software development

Table 1 presents several studies exploring the impact of collaboration on software development outcomes. Bettenburg et al. studied the impact of social structure on software quality and find a statistical relation between a communication flow between developers and users and post-release defects [6]. Abreu et al. investigated Eclipse sub-projects and found a significantly positive correlation between communication frequency between developers and number of injected defects in the software [1]. Bird et al. showed that a socio-technical network of software modules and developers is able to predict software failure proneness with greater accuracy than other prediction methods [7]. Wolf et al. formed a developer-task network to explore the impact of developer communication on software build integration fail [32]. Pinzger et al. constructed a developer-module network to predict the software failures [30].
More relevant to our focus are studies about the relationship between developer collaboration and defect fixing time. Feczak et al. empirically validated the Coordination theory in open source projects and found that collaboration among stakeholders, measured by social network metrics, has a positive influence on software defect fixing time [14]. Anbalagan et al. also found a significant correlation between the number of participants in editing a defect report and median time taken to correct it [2]. Guo et al. used collaboration measures to predict which defect will get fixed in Windows 7 and concluded that the defects that have more people involved in defect report editing will be more likely to be fixed [16]. While these studies show that developers collaboration, measured by a developer-artifact network metrics is useful for predicting software defects and fixing time, a similar approach can be applied to discover the impact of developers collaboration on issue resolution time.

2.2 Issue-Stakeholder network measures

Social network analysis (SNA) considers social relationships in term of network theories, which focus on social nodes, such as people, groups, organizations and measures relationships and information flows among them [15]. In this study, we construct an undirected graph to represent a network of issue-stakeholders. The graph employs two types of nodes: stakeholders and issues. Stakeholders include a reporter (who reports the issue), an assignee (who is assigned to resolve the issue) or a commenter (who comments or discuss about the issue). A link occurs only between a stakeholder and an issue, which represents for a stakeholder’s action on the issue, such as an issue report, a report update, a comment on the issue and an issue assignment.

![Figure 1: Issue-stakeholder network in issue resolution](image)

To establish the issue-stakeholder network, we use a social network analysis tool, namely ORA\(^\text{12}\). The most common measure in SNA is centrality, which denotes the structural power position of a node in a given network. There are three centrality measures in SNA, namely Freeman Degree Centrality, Closeness and Betweeness. In the scope of this study, we investigate Freeman Centrality Degree since this metric is successfully applied in relevant

\(^{12}\text{http://www.casos.cs.cmu.edu/projects/ora/}\)
studies [14][30][32]. In our network, the Freeman Degree Centrality of an issue represents the number of unique stakeholders that involve in the issue. For each issue, the high value of a centrality degree shows a large number of stakeholders working on it (reporting, commenting or resolving it).

The centrality degree of an issue is calculated as in Formula 1:

$$Gd(i) = \frac{d(i)}{n - 1}$$  \hspace{1cm} (1)

*with* $d(i)$ *is the node degree of a issue, n is the total number of stakeholders and issues*

<table>
<thead>
<tr>
<th>Table 1: Studies about collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bettenburg et al. [6]</td>
</tr>
<tr>
<td>Abreu et al. [1]</td>
</tr>
<tr>
<td>Bird et al. [7]</td>
</tr>
<tr>
<td>Wolf et al. [32]</td>
</tr>
<tr>
<td>Pinzger et al. [30]</td>
</tr>
<tr>
<td>Andrew et al. [5]</td>
</tr>
<tr>
<td>Feczak et al. [14]</td>
</tr>
<tr>
<td>Anbalagan et al. [2]</td>
</tr>
<tr>
<td>Guo et al. [16]</td>
</tr>
</tbody>
</table>

| 100 |
Similarly, the Freeman Degree Centrality of a stakeholder is the number of issues directly linked to the stakeholder. We also want to explore whether stakeholder centrality has an impact on issue resolution time. For each issue, we calculate the accumulative stakeholder centrality degree (Cs) as a sum of centrality degrees of all involved stakeholders, as in Formula 2:

\[
Cs(i) = Gdass(i) + Gdrep(i) + Gdcom(i) \quad (2)
\]

with \( Gdass(i) \), \( Gdrep(i) \) and \( Gdcom \) is the centrality degree of assignee, reporter and commenter correspondently.

The meaning of \( Cs(i) \) is that the issue is important when they are resolved by many stakeholders and by important stakeholders, who involved in many other issues. Illustrated by Figure 1, the Freeman centrality degree of Stakeholder 2 is 5/11 and the degree of Stakeholder 3 is 1/11, which shows that Stakeholder 2 involves in more issues than Stakeholder 3 does. Issue 3’s centrality degree is 3/11 and Issue 4’s centrality degree is 1/11, which shows that Issue 3 is involved by more stakeholders than Issue 4 is. The accumulative stakeholder centrality degree of Issue 3 is 7/11.

3 Research hypotheses

In our context, a firm-paid stakeholder is an assignee or a reporter who works for a commercial company that uses and contributes to the development of an OSS project. We observe that many firm-paid assignees are also core contributors in developing the OSS product. While these core project members have significant contributions in developing the software [12], it is interested to know whether they also significantly contribute to resolving issues in the software evolution phase. Therefore, our first hypothesis is that:

\( H1 \): The stakeholder’s centrality degree of a firm-paid assignee is higher than those of volunteer assignee. (Null hypothesis: there is no difference in distribution of stakeholder centrality degrees between firm-paid and volunteer assignees).

Since firm-paid assignees also include the core members of the projects, they are supposed to have more knowledge and experience in developing the OSS product than peripheral members do [12]. Therefore the resolution time should be different between the group of volunteer assignees and the group of firm-paid assignees. Our second hypothesis is that:

\( H2 \): There is a difference in mean issue resolution time between a firm-paid assignee and a volunteer assignee. (Null hypothesis: there is no difference in mean issue resolution time between firm-paid and volunteer assignees).

An issue with many stakeholders involved might relate to many different software modules or different development tasks. Therefore, the complexity of such issues is higher and thus, it takes the assignee longer time to resolve. Our third hypothesis is that:

\( H3 \): The larger number of stakeholders involve in an issue is, the longer the issue resolution time is. (Null hypothesis: there is no correlation between the number of stakeholders involved in an issue and the issue resolution time).
A large number of comments and discussions on an issue may be caused by problems on the issue description (which leads to confusion or dissensus among stakeholders) or by the complexity of the resolving task and could lead to longer resolution time. Our last hypothesis is that:

**H4:** The larger number of exchanged messages on an issue is, the longer the issue resolution time is. (Null hypothesis: there is no correlation between the number of message exchanged in an issue and the issue resolution time).

4. The case study

4.1. Projects Context and Selection

Three OSS projects were selected for our study, namely Qt, Qpid and Geronimo. The reasons for selecting these projects were: (1) these projects are active and ongoing for at least 4 years, which ensure the scale of the datasets; (2) there are similar issue tracking system used in these projects, which facilitate the data collection; (3) these projects are similar in business domain and technical level, thus reducing the variability of the results, and, (4) these projects are significantly influenced by firm-paid developers, which enable the investigation of the impact of different stakeholder types.

Qt is an Open Source cross-platform framework developed by Qt Development Framework (Nokia) based on the programming language C++. The framework offers common components such as networking, OpenGL, multimedia and a widget toolkit

Qpid is a cross-platform Open Source enterprise messaging system developed around the open standard Advanced Message Queuing Protocol (AMQP). It is implemented in many programming languages, such as: C++, C#, Python, Ruby and Java. The project originated from a joint venture mostly consisting of code by Red Hat, Iona and JP Morgan.

Geronimo is a server runtime framework that pulls together the Open Source alternatives to create runtime instances that meet the needs of developers and system administrators and open-source, Apache-licensed. The project originated from IBM developers.

4.2. Data collection and preprocessing

All software issues were collected from JIRA repositories of the respective projects. The summary of datasets was described in Table 2, with the main, owner firm of each project, the time frame of the issues collected for analysis, the total number of issues, number of stakeholders (assigned developers and issue reporters, who collaborated with the project during this period), the total number of issues in the

Issue resolution time was computed by using the created time field and the issue resolved

---

13 Qt project http://qt.nokia.com/
14 Qpid project http://qpid.apache.org
15 Geronimo project http://geronimo.apache.org/
16 JIRA–bug, issue and project tracking system, http://www.atlassian.com/software/jira/
time field. We excluded 3514 issues that are not possible to calculate the issue resolution time. We removed 2171 issues that have the state OPEN, DUPLICATE or INVALID. We also deleted 2838 issues that do not have reporter or assignee information (stated as unassigned or unknown), and issues with invalid stakeholder information (as described below). Twenty-two data points were also taken out by an outlier detection function implemented in the R\textsuperscript{6} package. At the end of the data preprocessing procedure, 16986 issues were selected for further analyses, which consume 67\% of total number of issues.

The classification of stakeholder type (firm-paid or volunteer) was manually executed by searching stakeholder name and professional information in the Internet. The first information source is the list of contributor and mailing list from the project repository. We found these stakeholders with explicit company information, either as project initiators or main contributors of the open projects. With stakeholders that company information was not given in the project site, we determined the affiliation by: (1) the stakeholder’s profile from social networking site such as Facebook, LinkedIn and personal blogs, and (2) the stakeholder’s email with a private company domain. The stakeholder company information were extracted by the time when the stakeholder worked in the OSS project. We assumed that the group of stakeholders (more than three) come from the same company participate in the OSS project as a company representative and are paid by the company. The stakeholders without any identified company information were classified as volunteers.

After collecting stakeholder information, we synchronized the stakeholder name and alias to avoid replicated data. Table 2 describes the total number of stakeholders that involve in the OSS projects in the time period that data are collected. Collaboration information was extracted from issue tracking systems and the mailing lists of OSS projects using a Perl script. For each issue, we collected comments, edits on the issue report and issue-related messages from the project mailing list.

4.3 Descriptive statistics

Table 3 presents the distribution of reported issues by stakeholder types in Qpid, Geronimo and Qt correspondingly. As our expectation, stakeholders from Redhat and JP Morgan in Qpid (53.6\% of reported issues) and stakeholders from IBM in Geronimo (60.8\% of reported issues) are the main contributors in reporting issues. However, the largest amount of reported issues in Qt comes from volunteer reporters (44.9\% of reported issues). This observation can be explained by the large amount of end-users involved in the Qt project, who directly report their problem, in the issue project tracking system. Table 4 shows the distribution of resolved issues by different stakeholder types. Not surprisingly, most of the issues are resolved by developers from the main firms such as Redhat and JP Morgan (62.4\% of resolved issues) in Qpid, IBM (71.6\% of resolved issues) in Geronimo and Nokia (62\% of resolved issues) in Qt.

Table 2: Issues collection from case studies

<table>
<thead>
<tr>
<th>Projects</th>
<th>Qt (Nokia)</th>
<th>Red Hat, JP Morgan</th>
<th>IBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Firms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Frame</td>
<td>11/03-12/10 (85 Months)</td>
<td>9/06-12/10 (51 Months)</td>
<td>8/03-12/10 (87 Months)</td>
</tr>
<tr>
<td>Number of Stakeholders</td>
<td>1568</td>
<td>126</td>
<td>405</td>
</tr>
<tr>
<td>Number of issues</td>
<td>16818</td>
<td>3016</td>
<td>5697</td>
</tr>
<tr>
<td>Number of selected issues</td>
<td>9921</td>
<td>2278</td>
<td>4787</td>
</tr>
</tbody>
</table>
Table 3: Distribution of Contribution in reporting issue

<table>
<thead>
<tr>
<th>Issues from</th>
<th>Qpid</th>
<th>Geronimo</th>
<th>Qt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td>453 (19.9%)</td>
<td>1205 (25.0%)</td>
<td>4452 (44.9%)</td>
</tr>
<tr>
<td>Other company</td>
<td>605 (26.5%)</td>
<td>683 (14.2%)</td>
<td>1124 (11.3%)</td>
</tr>
<tr>
<td>Main Firms</td>
<td>1220 (53.6%)</td>
<td>2919 (60.8%)</td>
<td>4345 (43.8%)</td>
</tr>
<tr>
<td>Total</td>
<td>2278 (100%)</td>
<td>4787 (100%)</td>
<td>9921 (100%)</td>
</tr>
</tbody>
</table>

Figure 4 presents box plot charts of issue centrality and issue-based messages in the three projects. In Figure 4a shows that most of issues are touched by one to three stakeholders, other than the reporter. In Figure 4b, the average number of issue-based messages is similar among three projects. We see that number of message exchanged around an issue in three projects is from none to four messages, slightly vary among projects.

Table 4: Distribution of contribution in resolving issues

<table>
<thead>
<tr>
<th>Issues from</th>
<th>Qpid</th>
<th>Geronimo</th>
<th>Qt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td>252 (11.1%)</td>
<td>401 (8.4%)</td>
<td>2463 (24.8%)</td>
</tr>
<tr>
<td>Other company</td>
<td>604 (26.5%)</td>
<td>956 (20.0%)</td>
<td>1315 (13.2%)</td>
</tr>
<tr>
<td>Main Firms</td>
<td>1422 (62.4%)</td>
<td>3420 (71.6%)</td>
<td>6143 (62.0 %)</td>
</tr>
<tr>
<td>Total</td>
<td>2278 (100%)</td>
<td>4787 (100%)</td>
<td>9921 (100%)</td>
</tr>
</tbody>
</table>

Figure 4a, b: Descriptive of issue centrality and issue-based messages

5 Hypotheses Testing Results

5.1 H1: The stakeholder’s centrality degree of a firm-paid assignee is higher than those of a volunteer assignee.

Due to the fact that stakeholder centrality degrees are not normally distributed as observed from histogram and descriptive statistics, we used Wilcoxon rank-sum test [13].

Table 5: Resolution time by volunteer vs. firm-paid assignees

<table>
<thead>
<tr>
<th>Projects</th>
<th>Median centrality of Firm-paid</th>
<th>Median centrality of Volunteer</th>
<th>Significance level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geronimo</td>
<td>0.0169</td>
<td>0.0049</td>
<td>p=0.0014</td>
</tr>
<tr>
<td>Qpid</td>
<td>0.0114</td>
<td>0.0057</td>
<td>p= 0.0251</td>
</tr>
<tr>
<td>Qt</td>
<td>0.0131</td>
<td>0.0024</td>
<td>p= 0.0014</td>
</tr>
</tbody>
</table>
All the tests are performed using the statistic package R with alpha = 0.05. The null hypothesis H1, which stated that there is no difference in stakeholder centrality degree between firm-paid and volunteer assignee was investigated with a one-tail test. The results are shown in Table 5. In all cases, the median values of centrality degree in the firm-paid groups are significantly higher than those in the volunteer groups. In particular, the number of issues involved by a firm-paid stakeholder is at least two times higher than ones involved by volunteer stakeholder in all projects. The p-values in all tests allow us to reject the null hypotheses in all projects. We accept the alternative hypothesis that the centrality degree of firm-paid stakeholders is higher than one of volunteer stakeholders.

5.2 H2: There is a difference in mean issue resolution time between a firm-paid assignee and a volunteer assignee.

The distribution of issue resolution time between firm-paid assignee and volunteer assignee is shown in Figure 5. From the graph, we notice that the difference between these two groups in Qt and Qpid is very small. In Geronimo, there is a slightly higher difference in distribution of issue resolution time between firm-paid and volunteer assignee, but the high standard deviation could make this insignificant. To test whether there is a difference in issue resolution time between firm-paid and volunteer assignees, we also used the Wilcoxon rank-sum test.

The null hypothesis H2, which stated that there is no difference in issue resolution time between firm-paid and volunteer assignee was investigated with a two-tail test. The results are shown in Table 6. We observed that in three cases, the test with Geronimo data revealed a significant difference in resolution time between two groups while those with Qt and Qpid data did not. Therefore, the null hypothesis was rejected only in Geronimo dataset at significance level 95%. In Qpid and Qt, we accept the assumption of the null hypothesis.

5.3. H3: The larger number of stakeholders involve in an issue is, the longer the issue resolution time is, and

H4: The larger number of exchanged message on an issue is, the longer the issue resolution time is
Table 6: Resolution time of volunteer vs. firm-paid assignees

<table>
<thead>
<tr>
<th>Projects</th>
<th>Median resolution time by Firm-paid</th>
<th>Median resolution time by Volunteer</th>
<th>Significance level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geronimo</td>
<td>10</td>
<td>18</td>
<td>p=0.0000</td>
</tr>
<tr>
<td>Qpid</td>
<td>23</td>
<td>17</td>
<td>p=0.1653</td>
</tr>
<tr>
<td>Qt</td>
<td>102</td>
<td>101</td>
<td>p=0.4911</td>
</tr>
</tbody>
</table>

We performed a pair-wise correlation analysis among number of message, issue centrality degree, sum of stakeholder centrality and issue resolution time. The correlation matrices for Qt, Qpid and Geronimo projects are shown in Table 7, Table 8 and Table 9 respectively. The mark “***” represents a significance level at 0.01. Referring to Hopskin interpretation of value of correlation coefficient, which classify the value of correlation coefficient as trivial (<0.1), minor (0.1 – 0.3), moderate (0.3-0.5), large (0.5 – 0.7), very large (0.7 – 0.9) and almost perfect (0.9 1.0) [22], the correlation between number of task-based messages and issue resolution time is significant at minor level in Qt, Qpid while it is at moderate level in Geronimo. The correlation between issue centrality and its resolution time is at a minor level for Qt and at a moderate level for Qpid and Geronimo. Besides, the correlation coefficient between stakeholder accumulative centrality and resolution time is slightly higher than the one of issue centrality. All of these correlation coefficients are significant at level 0.01, which allow us reject the null hypotheses for H3, H4 and accept the alternative ones. It is noticed that among three variables, the accumulative stakeholder centrality degree has the largest correlation coefficient with issue resolution time in all projects.

6. Discussion of Results

Table 10 summarizes the testing results for each hypothesis. Concerning hypothesis H1, the statistical test results reject the null hypotheses in all cases, which show the centrality degree of an average firm-paid assignee are significantly higher than that of an average volunteer assignee. This finding infers the distribution of labor between firm-paid and volunteer assignees. It indicates that in the issue-resolving process, a firm-paid assignee involves in much more issues than a volunteer assignee does.

On testing hypothesis H2, the issue resolution time significantly varies between firm-paid assignees and volunteer assignees in only one out of three investigated projects. Therefore, we can conclude that the stakeholder type is unlikely an influenced factor on issue resolution time. The data suggests that while volunteer and firm-paid assignees participate in OSS projects with different motivation and working approaches, these differences do not have an impact on their issue resolution time.

In the result for H3 and H4, the correlation tests reveal a positive correlation between collaboration measures, such as number of message, number of involved stakeholder and issue resolution time. It implies that the high collaboration level in an issue, e.g. high number of messages exchanged or high number of involved stakeholders indicates a longer resolution time. This may be due to the complexity of the task that relates other issues or software modules; or the poor quality of the issue description leads to demands of explanation and discussion. However, we are aware that the result of correlation analysis doesn’t imply cause-effect relationship due to the effect of compounding factors. To validate the provided hypothesis, a further regression analysis is necessary. From the results, we also observe that
there is significant positive correlation between issue centrality and number of messages exchanged. This observation was expected as the larger number of stakeholders involved in an issue (i.e. editing the reports or commenting on the issue) clearly leads to the increasing of number of comments or report edits. Therefore, these two variables should be checked for compounding factors if they are both used in regression models.

7. Threats to validity

First, a major threat of the study validity lies in the division of stakeholders as volunteer or firm-paid. Although a major amount of stakeholder’s affiliation is identified, there are still some stakeholders with no company information. However this group of unidentified stakeholders is responsible for a very small portion of issues in general. Since the major portion of the issues comes from identified stakeholders, the comparison of resolution time between groups of stakeholders would not be significantly influenced.

Second, a main concept investigated in this study is collaboration, which is measured by the number of comments, messages and number of issue-involved stakeholders. Although collaboration between stakeholders can be done via other channels, such as IRC, Skype and face-to-face discussion, issue tracking system and mailing list are the most common discussion means. The most relevant discussion about an issue should be found here. The other concern in the data collection process is the quality of the issue report since the data can be randomly filled in and the occurrence of duplicated reports. However, the quality of report is also an included factor in this study since it might influence the issue resolution time.

Table 7: Pairwise correlation for Qt

<table>
<thead>
<tr>
<th></th>
<th>No of message</th>
<th>Issue centrality</th>
<th>Sum. Stak. Centrality</th>
<th>Resolution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of message</td>
<td>1</td>
<td>0.413**</td>
<td>0.460**</td>
<td>0.125**</td>
</tr>
<tr>
<td>Issue centrality</td>
<td>1</td>
<td>0.213**</td>
<td>1.000**</td>
<td>0.172**</td>
</tr>
<tr>
<td>Sum. Stak. Centrality</td>
<td></td>
<td></td>
<td>1.000**</td>
<td>0.262**</td>
</tr>
<tr>
<td>Resolution time</td>
<td></td>
<td></td>
<td></td>
<td>1.000**</td>
</tr>
</tbody>
</table>

Table 8: Pairwise correlation for Qpid

<table>
<thead>
<tr>
<th></th>
<th>No of message</th>
<th>Issue centrality</th>
<th>Sum. Stak. Centrality</th>
<th>Resolution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of message</td>
<td>1</td>
<td>0.569**</td>
<td>0.423**</td>
<td>0.243**</td>
</tr>
<tr>
<td>Issue centrality</td>
<td>1</td>
<td>0.199**</td>
<td>1.000**</td>
<td>0.310**</td>
</tr>
<tr>
<td>Sum. Stak. Centrality</td>
<td></td>
<td></td>
<td>1.000**</td>
<td>0.331**</td>
</tr>
<tr>
<td>Resolution time</td>
<td></td>
<td></td>
<td></td>
<td>1.000**</td>
</tr>
</tbody>
</table>

Table 9: Pairwise correlation for Geronimo

<table>
<thead>
<tr>
<th></th>
<th>No of message</th>
<th>Issue centrality</th>
<th>Sum. Stak. Centrality</th>
<th>Resolution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of message</td>
<td>1</td>
<td>0.491**</td>
<td>0.382**</td>
<td>0.416**</td>
</tr>
<tr>
<td>Issue centrality</td>
<td>1</td>
<td>0.251**</td>
<td>1.000**</td>
<td>0.303**</td>
</tr>
<tr>
<td>Sum. Stak. Centrality</td>
<td></td>
<td></td>
<td>1.000**</td>
<td>0.409**</td>
</tr>
<tr>
<td>Resolution time</td>
<td></td>
<td></td>
<td></td>
<td>1.000**</td>
</tr>
</tbody>
</table>
### Table 10: Results of Hypotheses testing

<table>
<thead>
<tr>
<th>Test</th>
<th>Mann Whitney U</th>
<th>Mann Whitney U</th>
<th>Spearman correlation</th>
<th>Spearman correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geronimo</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>Qt</td>
<td>Accept</td>
<td>Reject</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>Qpid</td>
<td>Accept</td>
<td>Reject</td>
<td>Accept</td>
<td>Accept</td>
</tr>
</tbody>
</table>

Third, another threat to validity comes from the generality of the research findings. As in many empirical studies of OSS projects, few case studies are definitely not significant enough to generalize what we found to the population of OSS projects. In this study, the cases were thoroughly selected to represent for an active, medium-size and on-going OSS projects.

Last but not least, compounding factors is an unavoidable threat in a correlational study. The high correlation between number of messages, number of stakeholders and issue resolution time can be caused by a latent variable, not investigated in this study, such as complexity of the issue, or dependencies among issues. Therefore, this concern could be a subject for a future investigation.

### 8. Conclusion and future work

In this study, we investigated the impact of different stakeholder types and their collaboration on issue resolution time in three medium-size and ongoing OSS projects. The statistic test result provides some interesting findings for OSS practitioners as well as OSS researchers. First, we found that in firm-involved OSS projects, there is not only a large portion of firm-paid labor contributed to the projects, but also a higher workload on an average firm-paid assignee than on a average volunteer assignee. However, we did not find a difference in issue resolution time between volunteer and firm-paid assignees. The result contributes to the understanding of distribution of workload and resolving time between volunteer and firm-paid assignees. Second, we found a significant impact of stakeholder collaboration on issue resolution time. Particularly, the issue with fewer stakeholders is resolved faster than the one with more stakeholders. The issue with fewer comments is also resolved faster than the ones with more comments. For practitioners, these metrics can be integrated in the issue tracking system or defect repository to provide a recommendation for issue resolving process. Particularly, the collaboration information collected overtime will help developers being aware of which issue is going to take longer time to resolve. For researchers who want to integrate collaboration measures in software quality or productivity prediction models, they should consider of not only the usefulness of number of involved stakeholders, number of exchanged messages but also the compounding effect between them.

The paper contributes to fill in a gap in the literature gap by providing an empirical investigation of firm-paid stakeholders and their cooperation with others in OSS projects. The findings were supported by descriptive statistic and correlation analysis and further work should employ regression analysis to validate these findings. The study is also limited in using simple SNA metrics, such as the Freeman Centrality Degree. In future, we will explore more SNA metrics to investigate other aspects of stakeholder collaboration. Besides, the findings are based on only three projects, so the analysis should be replicated with more datasets to generalize conclusions on OSS community.
Acknowledgements

The authors would like to thank Tor Stålhane for his valuable comments and help with checking the statistical procedures.

References


Paper MP2 - Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components

Collaborative Resolution of Requirements Mismatches when adopting Open Source Components

Nguyen Duc Anh¹, Daniela S. Cruzes¹, Reidar Conradi¹, Martin Höst², Xavier Franch³, and Claudia Ayala³

¹ Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway
² Lund University, Department of Computer Science, Lund, Sweden
³ Technical University of Catalunya, Department of Service Engineering and Information Systems, Barcelona, Spain

Abstract. [Context and motivation] There is considerable flexibility in requirements specifications (both functional and non-functional), as well as in the features of available OSS components. This allows a collaborative matching and negotiation process between stakeholders such as: customers, software contractors and OSS communities, regarding desired requirements versus available and thus reusable OSS components. [Problem] However, inconclusive research exists on such cooperative processes. Not much empirical data exists supporting the conduction of such research based on observation of industrial OSS adoption projects. This paper investigates how functional and non-functional requirement mismatches are handled in practice. [Results] We found two common approaches to handle functional mismatches. The main resolution approach is to get the components changed by the development team, OSS community or commercial vendor. The other resolution approach is to influence requirements, often by postponing requirements. Overall, non-functional requirements are satisfactorily achieved by using OSS components. Last but not least, we found that the customer involvement could enhance functional mismatch resolution while OSS community involvement could improve nonfunctional mismatch resolution. [Contribution] Our data suggests that the selecting components should be done iteratively with close collaboration with stakeholders. Improvement in requirement mismatch resolution to requirements could be achieved by careful consideration of mismatches size, requirements flexibility and components quality.

Keywords: Requirements elicitation; Requirement mismatches; Open source software; Collaboration; Empirical study.

1 Introduction

The rapid growth in scale and complexity of software systems, together with the availability of third party software components, such as Commercial Off-The-Shelf (COTS) or Open Source Software (OSS) components, increase the adoption of component-based software development (CBSD) in software industry [20]. This adoption demands specialized software development processes that aim at supporting Off-The-Shelf (OTS, including both COTS and OSS) component acquisition, especially Requirements Engineering (RE) processes.

Traditional RE basically consists of eliciting stakeholder’s needs, refining the acquired goals into non-conflicting requirements statements, and finally validating these requirements with stakeholders [2]. The RE process for OSS-based development is quite different from this traditional one since integration with third party components is the essential part of software development. It is an intertwined process between requirements engineering activities and OTS component selection to select the best matched set of components and requirements. Therefore, requirements elicitation and negotiation becomes more likely a collaborative activity, which involves customers, software suppliers and third party vendors/communities. This collaborative process closely relates to the OSS component identification and selection.
processes [14]. The main challenge comes from the dynamic nature of requirements and evolution of OSS components [18, 26]. The continuously evolved requirements and updated versions of chosen components could make the component features differ from the requirements in post-selection phases. These mismatches between components and requirements are unavoidable and need to be resolved during the project lifetime.

Since the process of matching requirements and selected components is crucial for a successful adoption of OSS components in software projects, it is necessary to explore the relevant industrial collaboration practices, such as requirement elicitation, component selection and mismatch handling [6, 18]. Several studies have focused on the COTS component selection processes [1, 6]. However, less effort has been allocated to the investigation RE practices in the context of OSS component adoption and even less to empirical studies in this topic.

In this paper, we present a mixed quantitative and qualitative survey of how such requirement/OSS component selection and requirements mismatches are handled in fifteen European software-intensive companies in Norway, Sweden and Spain. The main purpose of the study is to explore the requirements and component selection practices and their relationships to the requirement-component mismatch resolution.

The remainder of the paper is organized as follows: Section 2 presents previous RE studies on COTS-based development. Section 3 describes our research approach. The results are provided in Section 4 and discussed in Section 5. The threats to validity and conclusions are given in Section 6 and Section 7.

2 Research background

2.1 Requirements-Components matching processes

Requirement component matching and mismatch resolving process are overlapping activities but occurs in different phases of CBSD. While component matching consists of eliciting requirements and finding matching components in early development phase [1, 4], mismatch resolution concerns about detecting the problems with selected components and resolving it in later development [5].

Literature reveals a significant amount of research on matching process [1, 3, 7, 16, 19]. Mohamed et al. summarized the evolution of COTS selection practices in 18 COTS selection approaches [1]. The common steps include defining the evaluation criteria using requirements, COTS search, filter search results, evaluation of COTS components, and selection of best-fit COTS. Stol et al. summarized 20 different initiatives for OSS component selection and evaluation [3]. Morisio et al. surveyed 15 COTS adoption projects and characterize the COTS adoption process [7]. The common steps for the requirement phase are requirement analysis, system requirements review, COTS identification and selection, glueware and integration requirement identification. The authors also found two major issues, namely dependence on the vendor and flexibility in requirements. Paech and Reuschenbach [19] present a requirements engineering process for OSS selection. In this process, the choice of product is based on a comparison of prioritized requirements from the stakeholders and evaluation results for candidate products. Höst et al. [16] summarize experience from a set of organizations on how to select open source components in software projects, and observe for
example that it is important to understand the requirements for the identified components.

These studies, nevertheless do not consider the dynamic nature of requirements as well as OSS components, which lead to the issues of requirement mismatches after selecting the best-fit component at the mentioned time.

2.2 Requirements-Components mismatches resolution process

Since component features are predetermined when selecting components, the changes in requirements introduce challenges to adoption of the components. A requirement-component mismatch is a difference in functional feature or non-functional quality attributes from a given component and a desired requirement.

On one hand, some studies see requirement negotiation as an approach to resolve the mismatches [2, 4, 5, 8, 10, 18]. In these cases, the component is fixed beforehand and requirements are the target of changes [18]. Maiden and Ncube observed that this process is iterative: from an initial stage with all the customer wish-list and the full marketplace available, mismatches progressively force requirements negotiation and candidate filtering until the final COTS component is selected [8]. Rolland proposed a goal-oriented approach for considering mismatches at the business level and then defined goal matching as the conceptual framework for resolving them [4]. Other approaches focused on lower level but highly challenging requirement problems, with integration requirements in call-for-tender processes [10].

On the other hand, a mismatch can be solved by modifying or adapting the selected components to fit to the requirements [5, 17, 26]. The components are modified when it takes a long time for external support [26] or when there is a need to adapt to new changes in requirement [5].

There is although a lack of empirical investigations of industrial practices on mismatch resolution. Consequently, there is no attempt to explore which approach is conducted in which scenario.

3 Research approach

3.1 Research questions

It is important to understand industrial practices on both requirement and component perspectives in order to investigate the mismatches between them in the later phases. The source of requirements and how they are described could infer how flexible the requirements can be. Besides, the component search and selection process could indicate potential problems with components while implementing requirements. The understanding of both perspectives leads to a comprehension of factors that influence requirement-component mismatches. This argument leads us to RQ1:

RQ1: What are the general practices of requirement elicitations and OSS component selection in OSS adoption software projects?

Secondly, we distinguish the concepts of functional and non-functional requirements with regard to requirement mismatches. In this study, we define functional mismatches as the
differences between functional requirements and features provided by the components. These functional mismatches are investigated in the component level. Since the functional requirements are often explicitly described, it is not problematic to identify the functional mismatches when they occur. We are interested in investigating how the functional mismatch between a requirement and a component is handled by project stakeholders. It is hypothesized as an intertwined process of negotiation and technical resolution that involve customer, developers and OSS community. To investigate this scenario in industry, we propose the RQ2:

**RQ2: How are the functional mismatches between requirements and OSS components collaboratively managed in OSS adoption software projects?**

Thirdly, in addition to discovering what functionalities are important to users at the system level, qualities associated with particular functionality/user goals should be elicited. The qualities may need to be translated by developers from user-level objectives, values and concerns into specific technical quality requirements, though nonfunctional requirements are often not well-described and poorly understood [22, 23], hence the mismatches between non-functional requirements and components are hard to investigate and assess. Besides, non-functional requirements are normally system characteristics. Therefore, they are often verified in the later phases of system development, when the modules are integrated and tested. Consequently, instead of investigating the mismatches between non-functional requirements and components, we investigated which and how non-functional requirements are fulfilled by using OSS components. This rationale leads to RQ3:

**RQ3: How are non-functional requirements fulfilled by using OSS components in OSS adoption software projects?**

### 3.2 Data collection and analysis

The study was performed in the period between September 2010 and September 2011, including study design, piloting, data collection and analysis.

![Figure 1: Research question mapping](image-url)
Population: Our target is software-intensive organizations that adopt OSS in producing software product. This population includes organizations with different sizes and in different application domains. 64 companies from our contact list were selected and contacted by phone call and email, in which fifteen stakeholders (developers or project leaders), who represented for 15 projects, agreed to participate in the survey. Some of the contacts were not eligible for participating due to several reasons, such as lack of adoption of OSS components in the projects, the companies changed the OSS adoption policy or the adoption strategy was not publishable.

Interview guide (survey): The method used in this study is semi-structured interviews. The interview guide was adjusted after three pilot interviews. The purpose of the survey is to discover the practices in OSS adoption, such as Requirements elicitation, Component selection, Requirement mismatch resolution and Collaboration process in adopting OSS components. In the scope of this study we focused on results extracted on RE practices. The survey was designed as a 5-section survey, with both closed and open questions. The closed questions were used to solicit information on interviewee and project context. The open questions were used to gather information on component-requirement mismatches resolution practices and communication to the community. The survey also included explanation for important terminology and description of context background in order to offer a common understanding for all participants. The relevant survey questions are given in the Appendix.

Data collection procedure: The interview survey was sent to all participants some days before the interview meeting. In this way, the participants could be well prepared for the interview. The participants were asked to fill in the first two parts of the survey and give back to us before hand. The next three parts of the survey were asked directly to the participant during the interview. Each interview session lasted between 40 to 75 minutes. Interviews were attended by one to three interviewers. The conversations were recorded and transcribed for posterior analysis. The transcripts vary from 13 to 21 pages in size.

Analysis procedure: We analyzed the filled-in questions and transcripts using a qualitative research tool NVIVO. The approach is a tailored thematic synthesis [12].

The analysis consists of four steps: extracting data from the interview transcription; grouping data into fundamental groups based on the structure of the survey; coding data within each category; translating codes into themes and linking relevant themes together. The first two authors examined the categories from different perspectives and searched for explicitly stated or concealed opinions about how Requirement Component mismatches are handled in industry. The results from the analysis are described in Section 4. For each research question, we conducted a quantitative summary of answers on closed questions from each interview and qualitative analysis of taped conversations to support the quantitative part.

4 Results

4.1 Projects description

We surveyed the requirement mismatch resolution process in fifteen projects from Norway, Sweden and Spain. Table 1 shows some of the projects characteristics of the surveyed projects. The team size ranges from two to 250 people. The project life cycles include ad hoc development, waterfall, iterative development and agile, with a prevalence of the agile model
in seven projects. The adoption of lightweight development life cycles, such as Agile or Scrum, introduces flexibility in requirements elicitation and component selection. The application domain covers a wide variety of domains, including Communication system, Information system, Web application and Public-sector support, with a dominant of Public sector support in five cases.

The OSS components portion represents the interviewees’ estimation about the proposition of actual use part of OSS components in total product size in LOC. The OSS portion ranges from 10 to 90%. In one project, the interviewee could not provide a percentage due to absent information about the total product size. The large portion of OSS shows the importance of OSS components in the software, which could influence the priority of components during the mismatch resolution process.

The last column indicates whether the component selection is decided in the RE phase or not. Interestingly, in seven projects, the components selection is not considered in the RE phase. In projects 5, 6, 13 and 15, the requirements are predetermined (i.e. subcontract or outsourcing) and selecting components are considered and design or coding level as an approach to implement given requirements. In Project 9, the company provides services to customers and selection of components is transient in RE phase.

Table 1: Projects characteristics

<table>
<thead>
<tr>
<th>ID</th>
<th>Team size</th>
<th>Development process</th>
<th>Application domain</th>
<th>OSS portion</th>
<th>Selection in RE?</th>
<th>RE source</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>20-25</td>
<td>Iterative</td>
<td>Communication system</td>
<td>90%</td>
<td>Yes</td>
<td>External</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>UNK</td>
<td>Audio/Video processing</td>
<td>10%</td>
<td>Yes</td>
<td>Internal</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>Agile</td>
<td>Search engine</td>
<td>80%</td>
<td>Yes</td>
<td>Internal</td>
</tr>
<tr>
<td>P4</td>
<td>18</td>
<td>Waterfall + Scrum</td>
<td>Embedded system</td>
<td>ca. 17000 KLOC</td>
<td>Yes</td>
<td>Internal</td>
</tr>
<tr>
<td>P5</td>
<td>2</td>
<td>Iterative</td>
<td>Oil/gas support product</td>
<td>77%</td>
<td>No</td>
<td>Internal</td>
</tr>
<tr>
<td>P6</td>
<td>200</td>
<td>Scrum</td>
<td>Public sector support</td>
<td>75%</td>
<td>No</td>
<td>External</td>
</tr>
<tr>
<td>P7</td>
<td>4</td>
<td>Scrum</td>
<td>Document processing</td>
<td>10%</td>
<td>Yes</td>
<td>External</td>
</tr>
<tr>
<td>P8</td>
<td>20</td>
<td>Agile</td>
<td>Public sector support</td>
<td>66%</td>
<td>Yes</td>
<td>External</td>
</tr>
<tr>
<td>P9</td>
<td>2</td>
<td>Agile</td>
<td>Information system</td>
<td>90%</td>
<td>No</td>
<td>External</td>
</tr>
<tr>
<td>P10</td>
<td>2</td>
<td>Iterative</td>
<td>Public sector support</td>
<td>60%</td>
<td>Yes</td>
<td>External</td>
</tr>
<tr>
<td>P11</td>
<td>250</td>
<td>Agile</td>
<td>Telecommunication</td>
<td>90%</td>
<td>No</td>
<td>External</td>
</tr>
<tr>
<td>P12</td>
<td>3</td>
<td>Ad-hoc, requirement-driven</td>
<td>University</td>
<td>90%</td>
<td>No</td>
<td>External</td>
</tr>
<tr>
<td>P13</td>
<td>3</td>
<td>Ad-hoc</td>
<td>Information system</td>
<td>5%</td>
<td>No</td>
<td>Internal</td>
</tr>
<tr>
<td>P14</td>
<td>5</td>
<td>Tailored waterfall</td>
<td>Public sector support</td>
<td>80%</td>
<td>Yes</td>
<td>External</td>
</tr>
<tr>
<td>P15</td>
<td>6</td>
<td>Iterative</td>
<td>Public sector support</td>
<td>20%</td>
<td>No</td>
<td>External</td>
</tr>
</tbody>
</table>
4.2 RQ1: What are the general practices of requirements elicitation and OSS component selection in OSS adoption software projects?

4.2.1. Requirements elicitation practices

**Source of requirements:** In eight projects, requirements come from external customers, and in one of the cases, managed by an external consulting company, as shown in Table 1. In one project the requirements come from both external customers and internal development team since customers required a system with similar functionalities of existing system. In this project, the requirements are flexible since the customers require the product to confront a predetermined standard and development team has to find out the detail requirements themselves.

In five projects, requirements are market-driven, coming from an internal development team. In three of them, developers also play the role of customers. Moreover, in the fourth one, they consulted other development teams that deployed similar systems, whilst in the fifth case the marketing department also had a stake. Another project’s requirements come solely from the marketing department. In this project, the software is a part of an embedded system to sale.

**Requirement description level:** Figure 2 shows that among investigated projects, seven projects have requirements coarsely described. We categorize the requirement specification according to three categories: coarsely, medium and detail based on requirement description and notation. The detail level of requirement specification iners the flexibility of the requirements since the coarser one is probably the more flexible one. The coarse description of requirements in major projects is probably caused by the adoption of agile methodology. Only three projects have requirements described in detail and three in-between. Concerning specification notations, free text is used as much as structured text. Both of these requirement notations are used in seven projects. Use cases and test cases are used in three projects each, and one case used “informal” flow diagrams for expressing navigational-related requirements in a web application.
4.2.2. Component identification and selection practices

Component identification: Figure 3 describes the approaches to identify the OSS components in company’s projects. Projects often used more than one approach. The most common approach is based on previous experiences without formal search and evaluation processes, which are used by ten out of fifteen interviewees. The second option (8 out of 15 interviewees) is either to use a search engine or to ask friends, colleagues or someone that has experience from before with the component. Both of these options were six interviewees mentioned about peer-review or grey literature as another source to find components. Only two projects contact customers during component identification process. One of the interviewees could not provide details in this questions nor the rest of this subsection since the selection process was entirely run by a team of software architects.

Component selection process: none of the interviewees reported the usage of formal evaluation processes, which are abundant in literature [1, 3], in their projects. This observation is similar to findings from a previous study [21]. The evaluation activity is normally undertaken in ad hoc manner. For small components, reading the documents or looking into the code is probably sufficient. For the more significant components, a survey may be conducted to search for alternative options. A short trial with the goal to “try to get it work as a proof-of-concept” is also one possibility.

4.3 RQ2: How are the functional mismatches between requirements and OSS components collaboratively managed in OSS adoption software projects?

4.3.1. Functional mismatches identification

Grounded from interview’s conversation, there are three main criteria used to decide on a mismatch between a requirement and an open source component, namely fit to functional requirements, fit to non-functional requirement and fit to legal requirement. As the basic purpose of using external components, the OSS components should have the basic functionalities that fit to the requirements. The functional mismatch is the ratio between part of the component that satisfies the requirement and the full set of requirement features. In case of small or fine-grained requirement (as in Figure 4a), the mismatch appears when there is a relative small portion of overlap functionality between the requirement and component. In case of large or coarse-grained requirement or product feature (as in Figure 4b), the mismatch happens when the component only provide part of required requirements.
With respect to non-functional requirements, reliability of the components is a highly cited criteria, and concerns the number of defects in the component; if the component is functionally fit to the requirement, but it contains many bugs then it would take a time and effort to use the components.

Last but not least, third criteria concern about component license issue. OSS components employ different types of licenses that would be taken into consideration, as one interviewee mentioned: “a lot of GPL license components cannot be used ... doing a mistake like shipping a GPL license component in a commercial product is very bad PR, and kind of legal problem...”.

### 4.3.2. Mismatches resolution approaches

Figure 5 provides the scenarios in which mismatches are handled. The majority answered that they change the components in some way, such as creating a glueware or addware, modifying the components and replacing the components, rather than get requirements affected. Nine interviewees said to modify or add adjustments to the OSS components by themselves. Six of them chose to make the changes globally, and send it back to the OSS community. Three interviewees make the changes locally, which are reserved for internal use only. Only two interviewees utilize community support for adapting the components while three interviewees chose commercial vendors instead.

### 4.3.3. When are requirements changed?

In most of the cases, the requirement is not a subject to change or relax as it is often at the higher priority over components. Some interviewees said: “... there is no case giving up on the requirements. Requirements are usually at first priority”, “... selecting an OSS component does not impact the requirement so much. It is not so much you can relax your requirement a bit or replace five hour of coding with existing component, it is not possible.” “... normally requirement is not in the position to relax it a lot.”, “... requirements were not negotiated because the project was about reengineering a legacy system into a web application; the requirements were the ones for the departing system”.

![Figure 4: Functional mismatch type](image-url)
Figure 5: Requirement component mismatch resolution approaches

In three projects, requirements come from predefined standards, government reform and they are not possible to negotiated or modified. In some other projects, the adopted components are of small to moderate size, and are implemented by domain specific libraries or as part of a framework. Since the integrated components serve for small and fine-grained functional requirement, it does not affect much on the overall requirements of the system. Some interviewees said: “requirements usually do not really affect choice of components that much, as most components we use are small and not visible to the customer”, “… we use smaller components rather than larger sort of application server or something, the customer doesn’t really see the component as a separate components, it is a part of the product”.

Besides, OSS components offer an opportunity to modify/adjust the components upon the mismatches. This flexibility of OSS components gives more chances to satisfy the requirements, as some interviewees said: “If there is a partial mismatch, I think we just use it for what we could use it for,”, “…was quite simple to extend the open source project to get the functionality we needed …”, “… one of the reasons to select one of the components was that it provides a proprietary script language that allows specifying its behavior when starting the system”. Particularly, in one project, the mismatched component was rewritten from the scratch since it was a small library.

We found only one case where the option of relaxing requirements was selectively taken. The development team adopted a compensatory strategy: whilst explaining to the customer which (non-critical) requirements were not satisfied, they emphasized additional functionalities that the OSS component was covering and could be incorporated into the delivered system. It was also helpful that the customer had a very technical profile and was able to understand the consequences (in terms of cost) of not relaxing the requirements.
4.3.4. When are requirements postponed?

While there is only one case where requirements is relaxed or modified, it is worth noticed that seven interviewees mention scenarios where some requirements were postponed. The requirements were postponed in some critical cases. In one case, requirements were postponed due to the quality of components: “...we have postponed the project because there are a lot of bugs in [Component name]. We have to look for a new library”. In the other case, the customer accepted to postpone some nonessential requirements, the strategy followed by the development team to convince the client was to highlight those features that were not required by the customer and were offered by the component.

4.4 RQ3: How are non-functional requirements fulfilled by using OSS components in OSS adoption software projects?

Figure 6 shows the perceptions of interviewee about non-functional requirements achieved by using OSS components. For each of non-functional requirement attribute, the grey column represents for the number of interviewees that mentioned about it. The black column shows the number of interviewees that satisfy with the quality attribute of the OSS component. As shown in Figure 5, the most concerned nonfunctional requirements regard to OSS components are performance, reliability, maintainability and cost. The list of concerned non-functional requirements in our study is different from the most concerned requirements in Berntsson Svensson et al., namely usability, performance and flexibility [24]. Their context was limited to the embedded system and market-driven projects and it may be the reason for the conflicting results.

4.4.1. Performance

Performance is satisfied by using OSS components in nine out of eleven interviews. The performance is perceived as sufficient or at least not affecting much the overall performance of the system. Some interviewees mention the problem with performance problems but these mainly come from hardware and infrastructure issues.

![Figure 6: Non-functional requirement fulfillment by OSS components](image)

123
4.4.2. Reliability

There are contradictory opinions about reliability of OSS component. Seven interviewees experienced good reliability, with little or few bugs, with the correctness of the system exceeding expectation. Four interviewees had experiences with both reliable and unreliable OSS components. There is a misunderstanding during the conversations with some interviewees between Reliability and Maintainability. Some people said the OSS component turned out to have sufficient reliability because the code is available and then it is easy to fix the bug.

4.4.3. Maintainability

Maintainability is an important feature for OSS components. Eight out of ten interviewees are satisfied with the maintainability of the components. The factors that that contribute positively to the maintainability of OSS components are:

- The openness of the code, that allows developers to “dive into the code” to fix bugs.
- Synchronization with the upstream development: the OSS community offer a chance for the company to escape from the burden of maintenance since the components can be synchronized with the upstream development, contact with the OSS community (in comparison with a commercial component), significantly influence the maintainability of the components.
- Documentation of the code that facilitate understanding and using components.

As maintainability is as important as reliability for selecting suitable OSS components, practitioners should look for components that are not only reliable but also has a high bug fixing rate.

4.4.4. Time and cost

Concerning time and cost, all of the interviewees are happy with the reduction of deliver time by using OSS components. Eight out of ten interviewees are happy with the cost due to the saving of licensing and implementation. There are two cases where cost is not satisfied. In one project, a lot of problems were reported due to the technical misuse of OSS component. At the end the team had governance problems that resulted in higher costs and poor reliability, performance and particularly maintainability, because the team in charge was not very big and the learning curve too steep.

5 Discussion

Our observations from fifteen projects with different context settings and requirement practices offer some implications for improvements in requirement mismatch handing process. The findings are consolidated in five propositions.

Proposition 1: market driven requirements are more flexible than bespoken requirements while resolving functional mismatches in OTS based development.

The result suggests that the choice of requirement mismatch handling approaches varies across projects and most likely do not depend on project context factors, such as: team size, application domain, development life cycle, portion of OSS components and component
selection phase. Therefore, the decision whether to modify OSS components or influence requirements is influenced by the nature of requirements and components themselves, e.g., type of requirement source. Among five projects with requirements from internal development teams, four of them have requirements postponed. The requirements from internal teams (or market-driven type of requirement) would be more flexible due to consideration of given functionality and implementation effort. The requirements from external customers (or bespoken type of requirement) are less flexible due to contractual predetermination in required functionality.

Proposition 2: A functional mismatch with a flexible requirement is resolved by postponing the requirement, rarely by changing it.

Although flexibility of requirement does not hinder the requirement priority, it is beneficial for mismatch resolution by extending the resolution time. Regardless of requirement source type, requirement is normally in the first priority. Therefore, the definition of requirement flexibility is associated with the ability to postpone requirements, rather than with the ability to change or give up on the requirements [7]. Postponing requirements often occurs with customer negotiation and debugging process.

Proposition 3: A small functional mismatch is resolved by modifying OSS component while a large functional mismatch is resolved by replacing it by another OSS component or a COTS one.

Our data suggests that the detail level of requirement and the size of components influence how mismatches are resolved. Given the flexibility of OSS components, the small mismatch (a fine-grained requirement with small component) require less effort to modify or rewrite while a large mismatch take much more effort to close the gap by adapting the components. This observation recommends that component selection in early phase, such as requirement elicitation, would be risky when the requirement is not clear enough and in general level. However, selecting components for finegrained requirements in later phase, such as design or implementation also have threats of extra cost in integrating small components.

Proposition 4: Component reliability issues lead to postponed requirements by fixing the component or replacing it.

Reliability is one of the most concerned non-functional attributes while adopting OSS component. It also receive contradict perception from interviewees. It is difficult to correctly evaluate component reliability in component selection phases. The information that are used as early quality indicators and selection criteria, such as number of fixed bugs, component reputation and project roadmap, is not sufficient. The problem in this non-functional attributes would influence functional requirements by delaying the accomplishment of these requirements. The fewer bugs in components would take more time to fix while many bugs in components would require for the replacement. In later case, the selection and matching process will be conducted again, which cost much more time and effort. This suggests a better care of nonfunctional requirements of OSS components when selecting components.

Proposition 5: A functional mismatch that gets support from the OSS community is associated with a perceived increase in satisfaction regarding component maintainability.
Three collaborative resolving requirement mismatch involve customers, OSS community and commercial vendor, alternatively. Keeping changes in components synchronized with OSS community is beneficial for fixing and maintaining these components. In resolving requirement mismatch, community involvement would not only reduce the developer’s effort in maintaining the components but also bringing more confidence on component quality as “given enough eyeballs, all bugs are shallow”. As maintainability is as important as reliability for selecting suitable OSS components, practitioners should look for components that are not only reliable but also has a high bug fixing rate.

6 Threats to validity

In this study, most variables are taken directly, or with little modification, from the existing literatures. To ensure that the given concepts are understood correctly by the interviewees, we sent the interview guide with a detailed description of the survey to the interviewee beforehand. One of the possible threats to the internal validity is our misunderstanding of respondents’ answers. Although at least two interviewers carried out the interviews and there was only one interviewee in each interview, we taped all interviews. Listening to the tape helped to ensure correct interpretation of answers and comments. However, having an independent (third) person to listen to the tape might increase data quality. During the interview, we tried to ensure the interviewee understand what they are asked. The primary threat to external validity is that the study is based on few and possibly not typical projects. In general, most empirical studies in industry suffer from non-representative participation. In the data sampling step, we tried to have projects with all sizes, from various domain application and have different portion of OSS adoption in the projects. Besides, this study is still a preliminary study. Future studies with more interviews will be implemented to give more statistically significant results.

7 Summary and future works

The main purpose of this study is to gain understanding of how requirements mismatches are collaboratively handled in OSS adoption projects. We found two scenarios in solving functional mismatches. The main resolution approach is to get the components changed by the development team themselves, OSS community or commercial vendor. The choice of adapting or replacing components depends on the mismatch size, component reliability and level of community support. The other resolution approach is to influence requirements, often by postponing requirements. This scenario is associated with issues of component reliability and maintainability. Non-functional requirements are satisfactorily achieved by using OSS components in general. Finally, we found that the customer involvement enhance functional mismatch resolution while OSS community collaboration could improve nonfunctional mismatch resolution.

The study identifies topics for future research on the requirement mismatches handling process. One of the potential future extensions of the study is a supporting framework for OSS component selection decision-making. The main purpose of the framework is to find out indicators of components reliability and maintainability from the OSS component community. Besides, some of the context factors show potential impact on requirement mismatch resolution decision, such as source of requirement or reliability of components. However, we do not have enough data to conduct a quantitative analysis on these factors. In future studies with more data points, a more quantitative analysis of impacting factors could be
implemented. Last but not least, we highlighted the importance of stakeholder involvement in mismatch resolving process. The deeper understanding of stakeholder involvement would help to improve the matching process.

Acknowledgements

This work has been supported by the Spanish project TIN2010-19130-C02-01 and partly funded by the Industrial Excellence Center EASE Embedded Applications Software Engineering, (http://ease.cs.lth.se).

Reference


Appendix

Part 1: Background Questions on Project and System (to be filled up prior to the meeting)

1.1 What was the mean annual staff-size of the project (both full and part-time employees)?
1.2 What part of the staff had previous experience with OSS-based development?
1.3 Did you have previous experience with OSS-based development before joining the project?
1.4 What was the total effort of the project?
1.5 What was (roughly) the starting time of the project?
1.6 What was the time of the first complete delivery from the project?
1.7 What were the major application domain(s) of the system?
1.8 Where did the requirements come from?
1.9 How were the functional Requirements described with regard to level of detail?
1.10 What was the overall, software development process/environment of the project?

Part 2: Identify initially some OSS Component candidates that may satisfy the Requirements

2.1 In which lifecycle phases were such OSS Components selected?
2.2 How was the search process and initial evaluation for such OSS Components done?
2.3 What were the main information sources in deciding whether the OSS Component candidates from point 2.2 could (partly) match your functional Requirements?

Part 3: Final evaluation and decision process to resolve possible Requirements mismatches vs. OSS Components

3.1 What did you do when the functional Requirements could not be sufficiently matched by OSS Component candidates?
3.2 How well were the major non-functional Requirements (“quality attributes”) achieved?
3.3 Focusing on the 5 most important functionalities from the Requirements, can you name and explain the matching OSS Components that you finally integrated into your system?
3.4 How big part of the system do the OSS Components now occupy?
Paper MP3 - OSS Integration Issues and Community Support: An Integrator Perspective

OSS Integration Issues and Community Support: An Integrator Perspective

Claudia Ayala¹, Daniela S. Cruzes², Anh Duc Nguyen², Reidar Conradi²
Xavier Franch¹, Martin Höst³, Muhammad Ali Babar⁴
¹ Universitat Politècnica de Catalunya  
{cayala, franch}@essi.upc.edu
² Norwegian University of Science and Technology  
{dcruzes, anhn, conradi}@idi.ntnu.no
³ Lund University, Sweden, martin.host@cs.lth.se
⁴ IT University of Copenhagen, Copenhagen, Denmark, malibaba@itu.dk

Abstract. The reuse and integration of Open Source Software (OSS) components provided by OSS communities is becoming an economical and strategic need for today’s organizations. The integration of OSS components provides many benefits, but also risks and challenges. One of the most important risks is the lack of effective and timely OSS community support for dealing with possible integration problems. For gaining an understanding of the common problems that organizations face when integrating OSS components, and the role played by OSS communities, we performed an exploratory study on 25 OSS integration projects from different European organizations. The results show that the main way of reducing integration problems was the use of OSS components from well-established communities; therefore very few integration problems were identified. In most of the cases these problems were successfully solved with the support from the OSS community and/or colleagues. In addition, contrary to the common belief that understanding code from someone else is a hard and undesirable task, some integrators consider OSS code even more understandable than their own code.

1 Introduction

The free availability of Open Source Software (OSS) has over the last decade had a significant impact, not only on the software IT industry, but also on software-intensive organizations. OSS is significantly influencing the ways these organizations develop, acquire, use, and commercialize software [1], and actual evidence shows that organizations are clearly becoming a very important part of the OSS communities. In particular, the integration of OSS components is one of the most popular ways of adopting OSS [2]. It involves including OSS components into other software products or systems and this again may involve modifying, extending, or wrapping the OSS components.

OSS integration might have many benefits, such as significantly lower (purchasing) costs, availability of high quality products, adherence to open standards and vendor independence [1]. However, it also implies several challenges. On the one hand, we may mention that OSS components do not always satisfy all the requirements. In certain cases, some “glue code” or modifications are required to make OSS components work together. This however creates a customized version of the OSS component. The integrator (i.e., the person(s) in charge of integrating OSS component(s) into the software system) is then faced with the issue of maintaining this derived version, and must decide how to handle these extensions and modifications. As a result, each organization that modifies OSS components and incorporates them in its own applications is faced with the issue of whether to contribute or not to the OSS community [3], [4]. On the other hand, some studies emphasize that high-quality OSS
components rely heavily on having a large, sustainable community to develop code rapidly, debug code effectively, and build new features [5]. Thus, the organizations that integrate OSS components into their systems represent a potential base of contributing members needed to sustain the OSS communities [6].

It is therefore vital to provide evidence that help OSS communities to envisage strategies to improve potential integration issues; as well as organizations to meet some practical challenges related to OSS integration. In this context, the goal for this study is therefore gaining an understanding of the common problems that organizations face when integrating OSS components, and the role that the OSS communities play in such integration processes. Thus, we conducted an empirical study on European organizations from Norway, Spain, Sweden and Denmark. It consisted on semi-structured interviews with 25 integrators from different organizations that represented 25 different integration projects. Based on their answers, we were able to draw some observations. We report our main findings in this paper.

The rest of the paper is organized as follows: Section 2 provides an overview of existing evidence on OSS integration and companies’ participation. Section 3 provides details of the empirical study. Section 4 summarizes the most relevant observations from the interviews. Section 5 discusses the results. Threats to validity are presented in Section 6, while Section 7 summarizes the conclusions and future work.

2 Background and Related Work

Recent systematic reviews reveal that integration is one of the most popular strategies of adopting OSS [2], [4]. The company-community relationships have been explored in works as [7], [8], [9]. In [7], the authors identify three types of organization-community relationships:
- Symbiotic: Both the community and the organization benefit from the relationship.
- Commensalistic: The organization benefits from the relationship but the community is not affected.
- Parasitic: The organization benefits from the relationship but at the same time it damages the community.

Several barriers to contribute back to the community have been also investigated as for instance by Ven and Mannaert [10] that found that deciding not to contribute can also be risky as one may be forced to maintain a parallel copy of the product. Furthermore, Stol and Ali Babar in [4] did a systematic synthesis of the reported challenges of integrating OSS and ended up with a comprehensive list of challenges related to OSS integration. Even though there is a considerable body of research on the challenges of integrating OSS components in the development of software products [2], [4], the majority of these works refer to success stories derived from single case studies or experience reports that provide very limited information about the real industrial landscape of companies integrating OSS components. Moreover, the role that OSS communities play on supporting integrators to solve their integration problems has not been further explored. Therefore, our overall objective is gaining an understanding of the common problems that organizations face when integrating OSS components, and the role that the OSS communities play in such integration processes.
3 Survey on OSS Integration Issues and Community Support

3.2 Survey questionnaire

Our overall objective has been broken down into two research questions that are at their turn broken down into more concrete sub research questions.

On the one hand, RQ1 was aimed to inquiry on potential integration issues. RQ1.1 is focused on inquiring the most common integration problems. Furthermore, the literature has pointed out the underestimation of integration effort and inefficient debugging as problematic areas that require further investigation [20]; therefore, we stated RQ1.2 and RQ1.3 respectively. Finally, as a previous study [13] reported that getting OSS components information seems to become a continuous monitoring activity rather than being on a project demand basis, we stated RQ1.4 to understand how integrators monitor OSS communities.

**RQ1: How do integrators deal with integration issues?**

- RQ1.1: What are the most relevant integration problems?
- RQ1.2: How are integration/testing costs estimated?
- RQ1.3: What are the differences on locating/fixing bespoke software bugs vs. OSS related bugs?
- RQ1.4: How are OSS communities being followed up?

On the other hand, as pointed out in the previous section, company-community relationships have been reported before (e.g., [7-9]), however, there are no sufficiently deep studies to further understand what kind of assistance and/or contributions are mostly requested/provided by integrators, and which means are used to do so. Therefore, RQ2.1, RQ2.2, and RQ2.3 were stated.

**RQ2: To what extent integrators interact with/contribute to the community?**

- RQ2.1-What kind of help do integrators request from the OSS community?
- RQ2.2-What kind of contributions do integrators provide to the OSS community?
- RQ2.3-Which means are used to interact with the OSS community?

3.2 Research Method

Interviews, observation and analysis of documents are some of the most common data collection methods. However, a stated in [12], qualitative (approached by interviews) and quantitative (approached by questionnaires) surveys are the two most relevant types of studies for component-based software engineering investigation. Thus, as the nature of our research questions was clearly exploratory, we decided to carry out the study using a qualitative research approach based on semi-structured interviews to collect data directly from software-intensive organizations that integrate OSS in software product development. Semi-structured interviews allowed us to have certain flexibility to further explore what was going on in the area.
Participants. Participating organizations were chosen from our direct or indirect industrial collaboration network. They include organizations with different sizes and in different application domains. 69 organizations were invited to participate by phone call and email. Some of the contacts were not eligible for participating due to several reasons, such as lack of integration of OSS components in the projects, or privacy of the OSS adoption strategy. We ended up with 25 integrators from different organizations that represented 25 different projects. Table 1 shows some details of the organizations and the analyzed projects.

The Instrument. The interview guide was carefully designed following the guidelines stated in [11] and previous experience performing international surveys from several members of the team [12], [13], [14], [15]. The survey was designed as a 5-section survey, with both closed and open questions. The closed questions were used to solicit information about the respondent and project context. The open questions were used to gather information on integration issues and community relationship. The survey also included an introductory section concerning relevant terminology and background in order to offer a common understanding to all participants. In this paper we report our finding related to the relationship among integrators and communities (other results from the study have been also reported in [16]). In general, the guide mostly focused on a single software development project with at least one release of the corresponding software product, and with integration of one or more OSS components. If the respondents had experience with several such projects, they were asked to choose the most familiar one.

Data Collection Procedure: The interview guide was sent to all participants some days before the interview meeting. In this way, they could be prepared for the interview. The participants were asked to fill in the first two parts of the survey and give back to us beforehand. The next three parts of the survey were asked directly to the participant during the interview. Interviews were mainly performed in the mother tongue of the respondents (some exceptions occurred in Norway, where the interviews were performed in English) and when possible face-to-face in their working place or by phone, by one to three researchers of the team. Interviews lasted around 40 to 75 minutes each and were recorded for subsequent analysis.

Data Analysis: Interviews were prepared for analysis by the manual transcription of audio records to text documents (the transcripts vary from 13 to 21 pages in size). When needed, a summary of each interview was translated to English so that the whole research team could assess and discuss the data. We analyzed the filled-in questions and transcripts using a qualitative approach that consisted on the assessment of the interview documents by two different researchers and the subsequent generation of categories by grouping sentences or phrases that described the same idea, action or property [11]. We tried to be exhaustive with the categories in order to include as much detail provided by the respondents as possible.

4 Results

This section presents the results of the study. They are grouped in 2 subsections according to the research questions introduced above, when possible, we use tables to illustrate the resulting categories.
4.1 RQ1: How Do Integrators Deal with Integration Issues?

RQ1.1 What Are the Most Relevant Integration Problems?
Twenty out of 25 respondents did not mention any relevant integration problem in the project they based their answers on. Some of them commented: “We use components that are like standards and with a big community behind, so it is hard that you are the first one that experiences a problem” (K); “In this case, we were lucky. The documentation was complete and updated” (P).

Only five respondents mentioned that they experienced some kind of integration problem. Two of them said that they dropped and changed the OSS component to solve the problem. One emphasizes: “[The potential problems] depend on getting the right component” (X). Two respondents agreed that the problem was solved by learning how other people proceeded in similar cases: “It was a problem related to incompatibility among versions. But, we solve it by searching in Google and finding people that have explained their solution for it” (O); “Yes, we had some problems, but they were already reported by someone else in the forum, so we just learn some tricks to solve it” (R). One respondent stated that the problem came from the lack of documentation “We struggle a little with data formats, because sometimes the documentation was incomplete” (V).

RQ1.2 How Are Integration/Testing Costs Estimated?
On the question about integration/testing costs, sixteen out of 25 respondents agreed that integration costs were estimated based on the experience of the development team. One respondent said: "There is a kind of guessing in this. We ask the development team and with their experience they come with numbers and we put a bill on it” (U). In addition, there were some mixed views on how costly the OSS integration was. One respondent for example thought the cost was low: “It is difficult to say, but in any cases it would be less than developing the component yourself. For the small component, the integration cost is very low anyway because they have a nicer interface…” (G). But another respondent said: “There is normally a lot of costs involved with testing and integration. Lots of money is involved from exchanging part to integrating part. Integration sometimes involves competition with closed systems or exchange with other systems” (Y).
Table 1. Some details of the organizations and projects studied

<table>
<thead>
<tr>
<th>Id</th>
<th># Employees</th>
<th>Application Domain</th>
<th>Project Staff</th>
<th>% of OSS exp.</th>
<th>Some OSS used</th>
<th>% OSS in the system</th>
<th>Total effort (person/months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>170</td>
<td>Defense (communications)</td>
<td>20-25</td>
<td>30%</td>
<td>JBPM, Jetty, Spring, LogBack, Maven</td>
<td>90%</td>
<td>&gt;2000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>ICT Industry</td>
<td>4</td>
<td>50%</td>
<td>Impact, LPng</td>
<td>10%</td>
<td>480</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>ICT Industry</td>
<td>2</td>
<td>100%</td>
<td>SolR, Xapian, Twisted: NLTK.</td>
<td>80%</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>350</td>
<td>Embedded systems</td>
<td>18</td>
<td>25%</td>
<td>Linux Kernel, MD5 Checksum</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>500</td>
<td>Oil and gas industry</td>
<td>2</td>
<td>50%</td>
<td>PDFLib, OpenPyExcel</td>
<td>77%</td>
<td>18</td>
</tr>
<tr>
<td>F</td>
<td>-</td>
<td>Public sector</td>
<td>200</td>
<td>60%</td>
<td>Flex Framework, Batch part of Spring</td>
<td>75%</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>230</td>
<td>Bank</td>
<td>4</td>
<td>100%</td>
<td>WideShot, CryptoPP, ParseXs</td>
<td>10%</td>
<td>36</td>
</tr>
<tr>
<td>H</td>
<td>190</td>
<td>Public sector</td>
<td>20</td>
<td>100%</td>
<td>JBoss, OpenSummer, USD</td>
<td>66%</td>
<td>1000</td>
</tr>
<tr>
<td>I</td>
<td>6</td>
<td>Finance</td>
<td>1.5</td>
<td>66%</td>
<td>Python, Soap and Django</td>
<td>90%</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>4</td>
<td>Public sector (Education)</td>
<td>3</td>
<td>100%</td>
<td>SunGridEngine, Cluster FS, Linux Debian, Ganglia</td>
<td>90%</td>
<td>30</td>
</tr>
<tr>
<td>K</td>
<td>100</td>
<td>Private services (entertainment, sales)</td>
<td>3</td>
<td>100%</td>
<td>Apache, MySQL, PHP, FFTP tools</td>
<td>5%</td>
<td>7.5</td>
</tr>
<tr>
<td>L</td>
<td>-</td>
<td>Public sector</td>
<td>5</td>
<td>100%</td>
<td>Mantis, Ant, Apache</td>
<td>80-90%</td>
<td>72</td>
</tr>
<tr>
<td>M</td>
<td>150</td>
<td>Public sector (Education)</td>
<td>6</td>
<td>100%</td>
<td>Jasper Reports, DOJO, Apache, Quark</td>
<td>25%</td>
<td>157</td>
</tr>
<tr>
<td>N</td>
<td>30</td>
<td>ICT</td>
<td>7</td>
<td>14%</td>
<td>Jenkins, Cucumber, Mercurial</td>
<td>10%</td>
<td>84</td>
</tr>
<tr>
<td>O</td>
<td>15</td>
<td>ICT</td>
<td>3</td>
<td>100%</td>
<td>Joomla</td>
<td>50%</td>
<td>56</td>
</tr>
<tr>
<td>P</td>
<td>5</td>
<td>Public sector</td>
<td>2.5</td>
<td>67%</td>
<td>Zope and Plone</td>
<td>99%</td>
<td>6</td>
</tr>
<tr>
<td>Q</td>
<td>14</td>
<td>ICT</td>
<td>3</td>
<td>100%</td>
<td>Varnish, Engine egg</td>
<td>80%</td>
<td>9</td>
</tr>
<tr>
<td>R</td>
<td>500</td>
<td>ICT</td>
<td>25</td>
<td>80%</td>
<td>Jasper Reports, Junit, Jmeter, MediaWiki, OpenCSV</td>
<td>30%</td>
<td>900</td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>Public sector</td>
<td>2</td>
<td>100%</td>
<td>RXTX, MySQL, Palcom</td>
<td>60%</td>
<td>36</td>
</tr>
<tr>
<td>T</td>
<td>&gt;1000</td>
<td>ICT</td>
<td>250</td>
<td>50%</td>
<td>OSS platform</td>
<td>50%</td>
<td>1000</td>
</tr>
<tr>
<td>U</td>
<td>11</td>
<td>Energy</td>
<td>2</td>
<td>100%</td>
<td>Speed -Typo3CMS, FPDF, Apache, Stability</td>
<td>40%</td>
<td>20</td>
</tr>
<tr>
<td>V</td>
<td>2500</td>
<td>ICT</td>
<td>4</td>
<td>100%</td>
<td>Mongo DB</td>
<td>100%</td>
<td>8</td>
</tr>
<tr>
<td>W</td>
<td>4</td>
<td>Whole-sale, retail and entertainment</td>
<td>10</td>
<td>50%</td>
<td>Ubuntu Enterprise Cloud (UEC) &amp; Eucalyptus, NappIt, pfSense, FreeBSD</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Public sector (Education)</td>
<td>1</td>
<td>100%</td>
<td>Zope, Plone, Apache, Mysql, Ubuntu</td>
<td>100%</td>
<td>3</td>
</tr>
</tbody>
</table>

Annotation: (-) respondent did not answer or asked to keep this information confidential
Table 2. Categories of Integration Costs Estimation

<table>
<thead>
<tr>
<th>Count</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Experience-Based</td>
</tr>
<tr>
<td>3</td>
<td>Did a pilot</td>
</tr>
<tr>
<td>2</td>
<td>In-house marketing department</td>
</tr>
<tr>
<td>2</td>
<td>Preliminary study of the candidate components and their integration problems</td>
</tr>
<tr>
<td>1</td>
<td>Testing tools</td>
</tr>
<tr>
<td>1</td>
<td>Templates</td>
</tr>
</tbody>
</table>

Three respondents pointed out that they used piloting as a way to estimate costs of integration. In these cases the pilot took from one to two months. Two respondents answered that the estimation was part of the preliminary study of the candidate components. In two interviews, respondents said that their organizations had a marketing department responsible for the estimation costs, so the respondents did not know details about such estimation. One respondent stated that their estimation was driven by a testing tool “We used a testing tool. Integration and testing was around 20% of the whole development” (Q). Finally, another integrator stated that they used specific templates for the estimation (T). Table 2 summarizes the obtained categories.

RQ1.3 What Are the Differences on Locating/Fixing Bespoke Software Bugs vs. OSS Related Bugs?

We inquired about the differences among bespoke vs. OSS bugs’ locating/fixing process. Nine respondents stated that they do not try to locate bugs in the OSS components. One of them commented: “the components we used are like standards. Everything has been proven several times and it is well documented, so we did not find bugs” (K). Nine respondents emphasize that there was no difference on how they located/fixed the bugs. At this respect, one respondent said: “in my experience, most open source libraries and components are well written and the author usually put pride in putting out something that is well commented and nice formatting, and usually it is quite easy to navigate around so; actually, the process is a bit similar” (E). On the other hand, two respondents said that the main difference resides on the fact that it is harder to look at someone else’s code. “We run code. If it does not work, we isolate the faulty areas. Then we get to know whether it is in own code or OSS code. It is usually in our own code. It rarely happens that OSS component has errors and they are cumbersome to resolve as we don’t know that code”(U).

One striking answer was on one respondent stating “It is harder to find bugs in our own code. In the OSS components we didn’t have the same amount of bugs than those bugs from us, because they were pretty much stable components. We didn’t have to do any formal testing in these OSS components” (F). One respondent stated that an external company was subcontracted to fix those bugs related to the OSS component that were not trivial “When there is a problem that is trivial or small, we try to fix it by our self. When the problem is something different from standard Linux libraries, we have a company to fix. It is a consultant that deals with third party libraries, mismatches…” (D). Finally, 3 respondents did not answer to this question. Table 3 summarizes the resulting categories.
Table 3. Differences Among Bespoke Software Bugs vs. OSS Bugs

<table>
<thead>
<tr>
<th>Count</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Do not try to locate OSS bugs</td>
</tr>
<tr>
<td>9</td>
<td>No difference with locating bespoke software bugs</td>
</tr>
<tr>
<td>3</td>
<td>No answer</td>
</tr>
<tr>
<td>2</td>
<td>It is harder to look at someone else code</td>
</tr>
<tr>
<td>1</td>
<td>Subcontract a company to fix OSS components bugs.</td>
</tr>
<tr>
<td>1</td>
<td>It is harder to find bugs in own code</td>
</tr>
</tbody>
</table>

RQ1.4 How Are OSS Communities Being Followed Up?

Fifteen respondents answered that they did not have someone following up with the OSS project. Some of their comments are: “No, only if there is a problem we go to the community” (J); “We don’t have anyone watching the update stuff…We don’t usually update the OSS component. For instance, now we chose the JBPM version 4.4. We wait sometime until someone realizes that there is a new version, but we don’t watch the community” (A).

Eight respondents stated that there was a responsible for OSS component issues. In seven of these eight cases, such a person was a colleague in the organization. One of them commented “Yes, there is a community coordinator who is the one that is the face of a community, and hence he/she follows the trends in this community.” One respondent stated that instead of having a dedicated person inside the organization, they subcontracted a company to select the OSS components and support them in any integration issue (D). Finally, two respondents did not answer this question.

4.2 RQ2: To What Extent Integrators Interact With/Contribute to the OSS Community?

RQ2.1 What Kind of Help Do Integrators Request From the OSS Community?

The analysis of the interviewees’ responses regarding the support from the community shows that thirteen respondents did not explicitly request help from the OSS community. Instead, they just used what it was already available on the community portal or managed to solve doubts by consulting their colleagues or using Google. "We did not make any contact extending the normal use of community forums and discussion boards. Most of our issues could be handled by information already available in the community portal” (F).

Ten respondents stated that for some specific aspects, they requested community help and were satisfied with the obtained support: "In a couple of technical aspects, we asked for opinions about what it was the better way to proceed" (P); “[There is] usually a very quick response” (E).

Table 4. Results of integrators’ contribution to the OSS community

<table>
<thead>
<tr>
<th>Answer</th>
<th>Own bug reports</th>
<th>Bug fixes with code</th>
<th>Become co-providers</th>
<th>Promoting the OSS culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>12</td>
<td>9</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NO</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>Unknown</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
One respondent stated that they started requesting help and became involved in the community so now they are active co-providers: “We were the ones that uploaded this part of the OSS, so we were the ones that better knew such part” (Q). Finally, one respondent stated that asked for help but did not get it (X).

**RQ2.2 What Kind of Contributions Do integrators Provide to the Community?**

We asked the respondents what kind of contributions they provided to the community. We consolidated their answers as shown in Table 4. On the one hand, twelve respondents stated that they had reported bugs, but only nine of them eventually contributed by fixing them. Some of their motivations were: “bug fixing is something we would sent back definitely because we are very interested to give it into the main branch so we don’t have to fix it every time we do an update” (H); “It is so much easier to get the bug fixed if you submit the fix of course. And with the open source project you can do that” (C).

On the other hand, twelve respondents stated that they mostly take advantage of the community without contributing: “we have not done anything. We just used the components” (N); “We do not dedicate a budget to OSS bugs notification nor contribution activities” (L). In addition, 4 respondents stated that they became co-providers of the community by contributing back some OSS components.

Finally, five respondents emphasize that they participate in organizations or activities to promote the OSS culture as for instance “We are founding members of Open source foundation” (U), or “We presented our resulting system in Workshops and Seminars to show how integrating OSS components can work” (J).

**RQ2.3 Which Means Are Used to Interact With the OSS Community?**

Nineteen respondents mentioned that they use to different extent bulletin boards, forums, email lists and the bug tracking system from the community project. Forums and bulletin boards were mentioned the most. However, there were six extreme cases were the respondents did not need any kind of direct interaction with the community: “No cooperation with community. We just downloaded the software” (V). “We did not need to communicate with the community as the components we used were very well documented” (K); “We don’t need direct contact with open source projects. We use the product because we have so much competence, either in the team or friend-to-friend. So, we don’t need to communicate with the community directly” (A); “We mostly read the documentation and things published in the OSS community, but did not collaborate directly. Furthermore, in cases when problems appeared, we used Google to find related hits or portals like StackOverFlow” (M).

5 Discussion of Main Findings

In this section we discuss the obtained results and establish whenever possible links to the findings of previous studies.

For most of the analyzed projects, integrators did not mention any relevant problem. Although this was an interesting observation, (as integration problems have been highlighted as one of the main concerns of organizations that integrate third-party components [20]), it is important to understand these results in the context of the analyzed projects. In fact, in the analyzed
projects, integrators tried to minimize potential integration problems by selecting OSS components that fulfilled an adequate level of documentation/information and/or ensuring that they would have enough (own or subcontracted) expertise to solve the potential problems. Thus, it can be observed from Table 1 that the OSS components used by most of the respondents refer to OSS projects with great activity and vitality. In addition, some of these OSS components have become de facto standards.

It is worth to mention that although some works have claimed that much of the literature does not reflect the huge diversity in OSS initiatives and projects, focusing instead on large, well-established communities. In our case, even if we did not have control over the projects selected by the organizations, we ended up mostly analyzing projects that integrated OSS components from well-established communities as organizations actually use these kinds of components. Thus, we agreed with Choi et al [18] that demonstrated that the mature status of well-known OSS projects likely attracts users given their greater activity and vitality. However this pathway is unavailable for most of the OSS projects and those newly initiated projects that struggle to attract users and contributors [19]. This also confirms the importance of studies that help OSS communities—especially those newly initiated projects that need to attract users to envisage strategies for attracting integrators.

Regarding the way bugs were processed, we found, on the one hand, that nine out of 25 respondents do not even try to locate bugs on OSS components; instead, they rely on the expected functionality. On the other hand, other nine respondents emphasized that it was not difference on the way they fixed bugs in their own code instead of fixing bugs from OSS components, mainly because the OSS code was understandable and well commented. In addition, one also said that OSS code is even more understandable than their own code. Most of them also claimed that finding bugs in OSS was not usual. In addition, it was interesting to see that 8 organizations have a responsible of the community trends. This seems to show the importance that the OSS communities are gaining in the organizations.

Regarding costs estimation, we found that most integrators did not further estimate integration costs; instead, they just made an informal approach based on their experience. So, it seems that the claim from Li et al [20] about the relevance of estimating the time that the component(s) integration takes, do not hold in most of our analyzed projects.

Furthermore, in most of the analyzed projects, integrators managed to deal with their integration problems by themselves, without requesting specific help to the community. They mostly used information/documentation already available in the community portal or asked their colleagues for help. In line with this observation, our results also show that forums and bulleting boards from OSS communities were typically used in a passive way (i.e., integrators navigated through documentation and previous posts more than actively participate by adding new posts or content).

It is worth to highlight that the perception of the integrators about the support received from the community was good. 24 out of 25 said that they managed to solve the potential integration problems by using the information available in the portal or requesting help to the community with usually a quick response. Only one case stated that he/she did not receive the expected help.

Regarding the integrator’s contribution, our results show that most integrators had limited interaction/contribution to the communities. This confirms the observations from [21-24] that
emphasize that most organizations seem to have rather limited contributions to the OSS communities. Furthermore, although our results show that the most frequent way to contribute was by providing bug reports without code, the number of integrators that also submitted the code for fixing the bug was also high. This seems to confirm the claim from [25] and [26] regarding that the number of organizations contributing to OSS seems to be increasing. In addition, other ways of contributing that have been usually overlooked by previous research are related to activities to promote the OSS culture by for instance funding OSS initiatives or sharing the knowledge with colleagues.

Regarding the involvement of the approached organizations in terms of the company-community relationships described by Dahlander and Magnusson [7], [8] (see section 2), our results show that almost all studied organizations seemed to have a commensalistic relationship with the community (i.e., the organization just benefits from the community). It was interesting to see that 4 out of 25 organizations have become active members of the community as co-providers of some specific parts of the OSS project, thus establishing a symbiotic relationship.

Furthermore, a common motivation for those that contributed to the community seemed to be to make sure that modifications to the component’s code were maintained, while a common inhibitor to contribute in those organizations that did not contribute was that their budget did not include time neither resources to participate in the communities. These factors have been also mentioned by Ven and Mannaert [10]. In addition, most integrators that did not contribute to the communities also mentioned that they try to use the component as is (i.e., without modifications). This agrees with the results stated by Li et al. [20] that showed evidence that the source code of OSS components is seldom modified, or Höst et al. [27] that in a focus group meeting found that practitioners based on their experience do not recommend adapting OSS components that are included in products. However, if they need to adapt them, the recommendation is to do this through “glue code”.

6 Limitations of the Study

This study was performed by means of a rigorous planning and the establishment of protocols for data collection and data analysis. This was especially important as the research involved several researchers and participants from different countries. In addition, the interview guide was carefully designed and piloted to improve its understandability. As a result, some changes in the interviews were done to enhance the elicitation process. Some vocabulary was defined at the beginning of the interview guide to homogenize concepts.

Some relevant decisions were taken for approaching a further understanding of the project contexts. One of these was to focus most of the questions of the interview guide on a single product development project so we could further inquire and analyze specific contexts of the projects. This enhanced the value of our analysis and observations. In addition, we sent the interview guide in advance to the respondents so that they could be informed of the kind of questions to be asked. As a result, when performing the study, we rarely experienced respondents having difficulty remembering project details. Furthermore, we explained to the respondents that our study was not focused on analyzing “wrong practices” but on knowing “how integration is done in industrial practice”. In several cases we experienced that the interviewer(s) shall skip some questions given time restrictions of the respondent; therefore,
some questions results did not cover all participants. Despite this, the results obtained for these questions were valuable as most of the respondents provide their answers. With respect to the data analysis strategy, recording all interviews (and later on transcribing them) contributed to a better understanding and assessment of the data gathered. The generated categories were analyzed, discussed and reviewed by all researchers of the team to ensure their accuracy, understanding and agreement.

Regarding external validity, we addressed several topics in our study. Some of the most relevant ones are listed. First, the companies in this study were selected by a strategy combining convenience and maximum variation sampling from 4 different countries (Spain, Norway, Denmark and Sweden). Second, we had no control over the projects chosen by the respondents. Nevertheless, most of the resulting projects from the participating companies did not cover domains such as real time or life critical requirements neither development for product lines. We are aware that these factors may have an impact on integration, and so we highlight that our findings should not be taken as assertions but also as potential hypotheses that need to be further validated. Thus, we emphasize that our results should not be generalized and might be interpreted with caution, keeping in mind the context from the participating organizations.

7 Conclusions

We have described the main findings from an exploratory study based on semi-structured interviews to integrators from organizations that integrate OSS components in their software products. The study aimed to explore the problems that organizations face when integrating OSS components, and the role that the OSS communities play in such integration processes.

The reported results might be valuable for researchers, organizations and OSS communities that may use the provided evidence to more clearly understand the real OSS integration problems that integrators face and properly align their efforts for facing them.

On the one hand, researchers may get an overview of the state of the practice, identify new research questions, and position and align their own work. On the other hand, organizations may use the provided evidence to understand how other companies integrate OSS and leverage their own integration strategy identifying the practical challenges they might face when doing so. Finally, OSS communities can be informed of the perception of integrators regarding support and to envisage improvements for fostering the collaboration of integrators with the community; this is especially useful for newly initiated OSS communities that usually struggle to attract contributors.

That is, researchers might need to establish new agendas or check potential hypothesis generated by our results. Practitioners might have to adjust processes or methodologies. And OSS communities might have to crate special integration groups or improve integration documentation.

Acknowledgments

We thank all people that participated in piloting an early version of the interview guide and the interview participants who took time from their workdays to participate in our interviews.
This work has been partly supported by the Spanish project TIN2010-19130-C02-01, and the Industrial Excellence Center EASE Embedded Applications Software Engineering, (http://ease.cs.lth.se). Jyoti Nandrajog contributed to the interview part of this study in Denmark.

References

Paper MP4 - The impact of global dispersion on coordination, team performance and software quality – A systematic literature review

The impact of global dispersion on coordination, team performance and software quality – a systematic literature review

Anh Nguyen-Duc, Daniela S. Cruzes, Reidar Conradi

Department of Computer and Information Science (IDI), NTNU, Sem Saelands vei 7-9, Trondheim, Norway

Abstract

Context: Global software development (GSD) contains different context setting dimensions, which are essential for effective teamwork and success of projects. Although considerable research effort has been made in this area, as yet, no agreement has been reached about the impact of these dispersion dimensions on team coordination and project outcomes.

Objective: This paper summarizes empirical evidence on the impact of global dispersion dimensions on coordination, team performance and project outcomes.

Method: We performed a systematic literature review of 46 publications from 25 journals and 19 conference and workshop proceedings, which were published between 2001 and 2013. Thematic analysis was used to identify global dimensions and their measures. Vote counting was used to decide on the impact trends of dispersion dimensions on team performance and software quality.

Results: Global dispersion dimensions are consistently conceptualized, but quantified in many different ways. Different dispersion dimensions are associated with a distinct set of coordination challenges. Overall, geographical dispersion tends to have a negative impact on team performance and software quality. Temporal dispersion tends to have a negative impact on software quality, but its impact on team performance is inconsistent and can be explained by type of performance.

Conclusion: For researchers, we reveal several opportunities for future research, such as coordination challenges in inter-organizational software projects, impact of processes and practices mismatches on project outcomes, evolution of coordination needs and mechanism over time and impact of dispersion dimensions on open source project outcomes. For practitioners, they should consider the tradeoff between cost and benefits while dispersing tasks, alignment impact of dispersion dimensions with individual and organizational objectives, coordination mechanisms as situational approaches and collocation of development activities of high quality demand components in GSD projects.

1. Introduction

GSD is a modern paradigm for developing software intensive systems [13, 31]. Software development and maintenance tasks are continually dispersed globally for cost saving, time-to-market shortening, technology innovation and operational efficiency [60]. A large amount of empirical research has been devoted to understanding general challenges [9, 32, 51], managing collaboration [18, 35], getting requirements [14, 15, 77], establishing trust [7], communication and coordination [34, 44] and engineering process [6, 25, 56] in GSD. Recent literature reviews of GSD literatures reveals team coordination in GSD projects as an important research area [18, 20, 26, 35, 51, 71].

It is commonly understood that the amount of coordination challenges in GSD projects depend on the extent the teams are globally distributed [18, 35]. However, studies on team coordination often have a lack of context description and the connection between different
dimensions of global context settings and team coordination. When talking about GSD, one often refers to team members from different geographical locations. However, other dimensions of global dispersion, such as temporal dispersion, cultural dispersion and organizational dispersion, are also visible in this context [21, 33, 62].

Empirical research on a relationship between global dispersion setting and project outcomes have shown different, event contradicting results on the effect size and its impact direction. For example, Herbsleb et al. (2003) investigated task resolution time in global software projects and found that geographical dispersion leads to inferior team performance [30]. However, a recent study by Kocaguneli et al. (2013) suggested that geographical dispersion has no impact on team performance [23]. While many contextual factors may explain the difference between these studies, the way the dispersion dimensions are defined and measured, might also play a role. A synthesis of these studies could provide a systematic view of commonalities and variations among them, as well as new interpretive explanations that go beyond the scope of any single study [12].

This study presents a systematic literature review (SLR) to understand different dimensions of global dispersion setting and how they impact team coordination and project outcomes. We adopted an input-process-outcome model to organize research questions and to provide the basis for integrating literatures [28]. As shown in Figure 1, the input element represents the starting conditions for teamwork, such as dispersion context of the projects. Although there is a large amount of GSD empirical studies, it is still not clear what and how dispersion dimensions are empirically investigated and measured (RQ1). With respect to process element, our study focuses on team coordination, as we are interested in understanding how dispersion dimensions create coordination challenges (RQ2). Via the team process factors, dispersion dimensions would have an impact on project outcomes. We would like to investigate how consistent these relationships are among primary studies in GSD (RQ3 and RQ4). Consequently, the research questions investigated in this systematic review are:

- RQ1: How are dispersion dimensions defined and measured in GSD studies?
- RQ2: What are the different coordination challenges that dispersion dimensions present to GSD project outcomes?
- RQ3: How do dispersion dimensions affect team performance in a GSD project?
- RQ4: How do dispersion dimensions affect software quality in a GSD project?

In order to decide if the systematic review shall be executed or not, a preliminary search on the topic was made. We searched Scopus and Google Scholar digital libraries, using the search string: (coordination OR collaboration OR cooperation) AND (software development OR software engineering) AND (systematic review OR systematic literature review). There were three relevant publications from Scopus and one relevant publication from Google Scholar found [18, 35, 71].

![Figure 1: Review framework](image-url)
Šmite et al. (2010) reviewed 59 empirical studies about the state of the art empirical studies in GSD [18], suggesting that the amount of GSE-related empirical studies is still relatively small and more investigation into cross-disciplinary research work is necessary. The authors discovered that a majority of empirical research represents problem-oriented reports focusing on different aspects of GSE management rather than in-depth analysis of particular practices or techniques. Our study is different from this review in its research objective and search scope, as we want to establish relationship between dispersion dimensions, coordination and project outcomes. Learning from Šmite’s review, we looked at papers from different disciplines (see Section 3.6.1). One of our main findings is that temporal dispersion tends to be perceived more pessimistically than its actual impact.

Noll et al. (2011) reviewed 26 studies about challenges and solutions for collaboration in global software development teams [35]. The key barriers to collaboration are identified as geographic, temporal, cultural, and linguistic dispersion; and also solutions to overcome these barriers, such as site visits, synchronous communication technology, and knowledge sharing infrastructure. While this study performs a narrative summary of collaboration in distributed context in general, we systematically categorized dispersion dimensions based on their concepts and measures. These dimensions are then linked to coordination challenges and project outcomes.

Steinmacher et al. (2010) reviewed 42 studies about awareness support in GSD projects and its impact on team communication, coordination, and cooperation [71]. The authors classified awareness factors that affect team collaboration. The review concluded that coordination is the most explored dimension, while awareness support in communication is very poorly studied. Our study differs from this review in the way that we synthesized team coordination challenges created by different global context factors. Additionally, Steinmacher et al. included both empirical, theoretical and tool review papers, while our review only focuses on empirical evidence.

In a prior submitted version of this study, we reported the relationship of dispersion dimensions, coordination mechanism and their impact on team performance [55]. In this paper, the extended contributions are:

- A synthesis of the relationship between dispersion dimensions and software quality
- A new search iteration to update relevant GSD studies in 2012 and 2013
- An in-depth insight on the contradictory results on the impact of dispersion on team performance
- A set of propositions and future research direction for researchers in GSD
- A set of recommendations for practitioners to configure GSD teams

The paper is organized as follows: Section 2 introduces main concepts on dispersion dimensions and team coordination. Section 3 describes our review and synthesis approaches. While Section 4 shows our results and synthesis, Section 5 is discussion. Finally, Section 6 summarizes the findings and also presents conclusions of the paper.
2. Main concepts

**Global software project** has its activities, i.e. eliciting requirements, design architectures, implementation and maintenance that occurred in a dispersed environment. A project is dispersed in different scenarios, such as: offshoring (relocation of software production and services from one country to another [2, 69]), outsourcing (contracting out software development tasks or processes to an independent organization [1]), onshoring (leveraging resources from the same country (probably from different companies) [19]), nearshoring (outsourcing to a country closer in geographic, temporal, cultural. i.e. Japan and China; Germany and Belarus [22]). Open sourcing is also a type of globally distributed development, where commercial firms utilize software components from contributors (volunteers or commercial firms) distributed around the world [78].

**Distributed software team** is defined as “groups of geographically, organizationally and/or time dispersed workers brought together by information and telecommunication technologies to accomplish one or more organizational tasks” [62]. A software team can include members from different organizational units and participate in the same software development (or maintenance) tasks.

**Global dispersion dimensions** are different context factors of a global software project that can influence its outcomes:

- Geographical dimensions occur when project members are located in different physical places, from which it is not possible for them to perform direct communication, as they do not work in the same room [28, 18, 35, 62, 33, 21 55, 6]
- Temporal dispersion occurs when there is a difference in working time among distributed team members [28, 18, 35, 55, 62, 33, 21, 6]
- Work dispersion refers to variations in working procedure issues, such as organizational and personal work process, management policy and control [55, 42, 59]
- Cultural dispersion is the difference in team members’ languages, cultural backgrounds and national contexts [28, 18, 35, 55, 33, 6]
- Organizational dispersion occurs when a team consists of members from different organizations or independent organizational units [33, 53, 54]

**Software metrics:** provides continuous measures for the software development process and its related products [80]. Software metrics are categorized into three main groups: (1) product metrics involving measurement of different features of documents and programs generated during the software development process, (2) process metrics related to measurement of activity that happened during the software development life cycle, such as software design, implementation, test, and maintenance and (3) resource metrics measures supporting resources such as programmers, and the cost of the product and processes [81]. In this review, we focus on process and resource metrics that are proxies of global context settings in GSD projects.
**Team coordination** is defined as “activities required to maintain consistency within a work product or to manage dependencies within the workflow” [5]. There are many different types of dependencies between task and task holders, such as shared resources, task assignments, simultaneous constraints and task – subtask relationship [75]. These dependencies lead to a need for coordination among stakeholders working on related sets of tasks. When these coordination needs are not satisfied, there will be coordination challenges [30, 34].

**Project outcomes** can be seen from different viewpoints, such as: personal levels like personal learning, personal performance, and team level, as well as team productivity [29], product outcomes, such as quality attributes (reliability, performance, security, etc.) [70] and (3) organizational levels, such as process improvement, project revenue, customer satisfaction and company portfolio [27]. In this review, we focused on the outcomes at the team and organizational level. The evaluation of project outcomes can come from: direct measure of task duration or number of defects (objective team performance), or be based on the perception of team members and other stakeholders, i.e. project managers’ and customers’ opinions about team performance and product quality (perceived team performance).

### 3. Methodological approach

We planned, conducted, and reported review findings by following the SLR process suggested by Kitchenham (2007) [3]. Our study is characterized as a mapping study, as we classified measures of dispersion dimensions (RQ1) and also coordination challenges (RQ2). Moreover, our study is characterized as a systematic review, as we conducted a type of meta-analysis and provided insights on primary studies (RQ3 and RQ4) [40]. In regards to the scope of study, we considered outcome and quality assessment of the articles as a major focus, so increased the depth of the study, rather than the breath. The complete review lasted for 23 months, including six main steps, as described in Section 3.1.

#### 3.1. Review procedure

**Step 1: Ad-hoc review**

We conducted an ad-hoc literature review on team coordination in GSD. The goal was to understand conceptual background and emergent issues for further investigation. We started with a small set of high quality and relevant articles suggested by experts in the field. A snowball search with forward and backward reference scanning was performed. This step resulted in 27 papers. This initial pool was used to formulate the search string and to perform a validity check for the systematic search later.

**Step 2: Pilot search**

We developed a review protocol (as described in Section 3.2), which specifies search terms, databases, search inclusion and exclusion criteria, quality assessment and a data extraction form [70]. The pilot search and selection was conducted via Scopus to refine the review protocol (inclusion, exclusion criteria, and quality assessment checklist). The selection pool was limited to papers published in 2005 (980 papers), and was used to conduct a pilot data synthesis. This set of papers was also used to compare against the results of an actual selection.

**Step 3: Systematic search and extraction**
The actual systematic search was performed using search protocol developed from previous steps. The result of this step was a complete pool of 11,222 unduplicated papers from Scopus. By checking papers by titles and abstracts, we limited it to 470 papers. After checking papers by reading the full text, we selected 73 relevant papers, as shown in Table 1. Relevant information was extracted from these studies using the data extraction form described in Section 3.3.

**Step 4: Additional manual search**
A manual search was performed to retrieve more relevant papers and to search for gray literature. The reference list of selected papers was scanned in order to find more relevant papers. The manual search in related conferences and journals was also conducted. In the end, the additional manual search in Google added some extra papers that are not indexed by Scopus, IEEE Explore or ACM Digital library or were not published. In the end of this step, we identified 20 extra relevant papers. After quality assessment, we added three papers to the final set (S22, S34 and S42).

**Step 5: Quality assessment**
Paper quality assessment was conducted to filter out low quality papers (details in Section 3.4). In this step, we also removed papers that are identical in data set, investigated dependent and independent factors and the level of analysis. After that, we identified 42 relevant studies. Twenty-eight studies from this list, which reported the effect of GSD development on performance, were used in a conference version of this study [55].

**Step 6: Data analysis**
After relevant information from primary sources was collected, we conducted a multi-step analysis to answer RQs. The analysis approaches are described in Section 3.5.

**Step 7: Additional search**
After publishing the preliminary findings in ESEM conference, we conducted a search process again to update it with new relevant studies from 2012 to 2013. We also rescanned papers in 2011, which might not be updated yet in Scopus and hence, escaped from the previous search. At the end of this step, three additional relevant papers were found.

### Table 1: Systematic review procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Purpose</th>
<th>Time frame</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad hoc review</td>
<td>Conceptual background and emergent issues</td>
<td>05/11-09/11</td>
<td>27</td>
</tr>
<tr>
<td>Pilot search</td>
<td>Refine search protocol</td>
<td>10/11-12/11</td>
<td>980</td>
</tr>
<tr>
<td>Systematic search and</td>
<td>Identify all studies</td>
<td>12/11-02/12</td>
<td>11222</td>
</tr>
<tr>
<td>extraction</td>
<td>Select by title, abstract</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select by full text</td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>Manual search</td>
<td>Find grey literature</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Quality assessment</td>
<td>Filter low quality paper</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>ESEM paper 2012</td>
<td></td>
<td>03/12</td>
<td>28</td>
</tr>
<tr>
<td>Additional search</td>
<td>Update with paper from 2012 and 2013</td>
<td>03/13-04/13</td>
<td>3</td>
</tr>
<tr>
<td>Final set</td>
<td></td>
<td></td>
<td>45</td>
</tr>
</tbody>
</table>
3.2. Review protocol

3.2.1. Search terms:

Derived from the main research question, the search string consisted of 3 parts: (Coordination Or synonyms) AND (Dispersion Or synonyms) AND context. The synonyms were identified in the context of SE (Software Engineering) and Information System (IS). As the ad-hoc review revealed that there was no distinguished use of terms “coordination,” “collaboration,” and “cooperation,” in SE and IS literature, we used them as interchangeable concepts. We also found that GSD occurs in many different contexts and is described by various terms. Therefore, we tried to cover the terms represented for GSD from the ad hoc review and the GSD terminology list suggested by Šmite et al. (2010) [18]. Consequently, a list of related terms was identified: distributed, distribution, dispersed, dispersion, remote, offshoring, offshore, outsourcing, outsource, nearshore, and global.

A trial search with these terms resulted in many irrelevant studies, which came from other fields, such as business, organizational science, human studies and robotics. To limit the number of relevant studies, we tried to add some desirable context characteristics of studies, such as empirical studies and software development. The search terms were adjusted based on the search coverage and accuracy [57]. We validated the quality of the search string with the favor of coverage to accuracy [57]. That means we wanted to have a search string cover as much as possible. We conducted a several trial searches for papers in 2005 and compared the result with the initial pool. The best coverage is 85.70% and accuracy is 0.47%. The final search string was:

(coordinati* or collaborativ* or cooperati* ) AND (distributed or offshor* or "open source" or outsourc* or global or dispers*) AND (software or project or team) IN (Title or abstract or Keyword)

3.2.2. Search database:

A review through systematic reviews in SE showed several choices for electronic database, such as Scopus, ISI Web of Science, IEEE Explore, Current Contents, Kluwer Online, Computer Database, Science Direct, Springer Link, Inspec and ACM Digital Library. We decided to select an index database that has good coverage, familiarity, reputation, advanced features and exportability [48]. Scopus and ISI Web of Science were selected due to previous experiences of reviewers, flexible formulation of search strings with unlimited clauses and being able to easily export paper lists in various formats.

3.2.3. Inclusions and exclusions criteria:

We included papers that:

- Reported studies in the context of global software development context (including SE and IS literatures)
- Explored one of the global dispersion dimensions and related them to either team coordination or a type of project outcomes
- Adopted one of the empirical methods: case study, observation, controlled experiment, report data or empirical analysis
Both studies conducted in industry and in academic environment were considered. We also included both qualitative and quantitative studies. The limitation was that we only searched for papers written in English. The studies were excluded if they did not help to answer at least two research questions. We excluded papers that:

- Investigated team performance or quality in global context, but did not explicitly address the relationship between coordination or global dispersions and project outcomes.
- Investigated team coordination in other domains, such as business, education science and system engineering.
- Proposed coordination-supported tools, such as: wiki-based systems, social network-based platforms.
- Provided a conceptual framework or model about coordination that results from literature reviews or author’s thoughts without empirical validation.
- Had the same set of data, with identical dependent and independent variables, and the same level of analysis.

The extracted studies were also excluded from the final set if they did not pass the minimum quality scores. During pilot review, we considered whether to select short papers, which pass inclusion criteria (less than 6 pages). Given the main review objective is to synthesize relationships between dispersion dimensions and project outcomes, we selected papers that had a proper research design with sufficient descriptions of concept, measures of independent and dependent factors. Consequently, most of the short papers would not pass a quality assessment due to a lack of sufficient descriptions of research approaches [32, 65, 43, 58, 39, 49, 72]. Therefore, we decided not to select short papers during the systematic search and selection.

3.3. Data extraction

The data extraction was done by the main author and the extraction result was discussed with other coauthors. The extraction form was constructed by a mix of top-down and bottom-up approaches, as given in Table 2 (details in Appendix 1).

<table>
<thead>
<tr>
<th>Category</th>
<th>Property</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study background</td>
<td>Study meta data</td>
<td>Demographics</td>
</tr>
<tr>
<td>Study design</td>
<td>Study design</td>
<td>Demographics</td>
</tr>
<tr>
<td>Study context</td>
<td>Context settings</td>
<td>RQ2, RQ3, RQ4</td>
</tr>
<tr>
<td>Independent factors</td>
<td>Dispersion dimensions definitions and measure</td>
<td>RQ1</td>
</tr>
<tr>
<td></td>
<td>Team coordination definitions and measure</td>
<td>RQ2</td>
</tr>
<tr>
<td></td>
<td>Other control factors</td>
<td>RQ3, RQ4</td>
</tr>
<tr>
<td>Dependent factors</td>
<td>Project outcomes definitions and measure</td>
<td>RQ3, RQ4</td>
</tr>
<tr>
<td>Findings on the relationship</td>
<td>Relationship between dispersion and team coordination</td>
<td>RQ2</td>
</tr>
<tr>
<td></td>
<td>Relationship between dispersion and project outcomes</td>
<td>RQ3, RQ4</td>
</tr>
</tbody>
</table>
Firstly, we constructed a general form with data fields obtained from the ad hoc review [50, 73]. After that, fields in the extraction form were iteratively refined during the pilot data extraction. The refinement and adjustment continued until we reached saturation.

**Meta data:** Information about author’s name, publication year, and publication sources were collected for demographics of primary studies. We also collected the texts that described key concepts in the study background. This information helped us to understand more about the dependent and independent factors in the study.

**Study design:** we collected information about sampling strategy, unit of analysis, sample size, data collection methods and data analysis methods. Sample size and research design quality showed the strength of evidence provided by the study. For unit of analysis, we wanted to see if the data is collected and analyzed at task, team or project level. Study subjects included students, programmers or managers. The research methods were one of the following: experiment, case study, ethnography, survey, interview and archive mining. We distinguished studies that investigated multiple data sources in a period of time (case study) with studies that explored project repository as the only data source. Study analysis methods could be qualitative or quantitative. Qualitative analysis involved descriptive and interpretative analysis of interviews or observation data while quantitative analysis included statistical test, correlation analysis and regression model of data [10, 46].

**Context setting** consisted of descriptions of product type, project distribution type, number of distributed sites, site locations, adopted coordination technology and practices. We determined this from among four types of project distribution. The project could be distributed among sites within a single organization (intra-organization), or among more than one organization (inter-organization). The projects could be open-source projects or they could occur in laboratory settings. We also collected information about the number of involved sites and site locations as they implied detailed descriptions of dispersion dimensions.

**Independent and dependent factors:** We collected the text that describes dispersion as a property of the team. In quantitative study, we collected definitions and measures of dispersion dimensions, while in qualitative study we gathered descriptions and perceptions on dispersion dimensions. We also collected texts describing either team coordination problems or team coordination practices. Finally, the definition and measures of project outcomes were collected.

**Findings on the relationship:** We identified and extracted texts that reported relationships between dispersion dimensions, team coordination and project outcomes. It was not necessary that all of primary studies reported both of these relationships. From these texts, we performed a synthesis, as described in Section 3.6, to answer RQs.

**3.4. Quality assessment**

Paper quality assessment was performed to evaluate the relative strength of empirical evidences reported. We devised a number of quality assessment questions to assess the rigorousness, credibility, and relevance of the relevant studies from Dybå and Dingsøyr’s checklist [73], as shown in Table 3.
Table 3: Quality assessment checklist

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Research design</th>
<th>Research design</th>
<th>Research design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1. Is research objective sufficiently explained and well-motivated?</td>
<td>Q2. Is the context of study clearly stated?</td>
<td>Q3. Is the research design prepared sufficiently?</td>
<td>Data collection</td>
</tr>
<tr>
<td>Data collection</td>
<td>Q4. Are the data collection &amp; measures adequately described?</td>
<td>Q5. Are the measures and constructs used in the study the most relevant for answering the research question?</td>
<td>Data analysis</td>
</tr>
<tr>
<td>Q6. Is the data analysis used in the study adequately described?</td>
<td>Q7a. Qualitative study: Are the interpretations of evidences clearly described?</td>
<td>Q7b. Quantitative study: Are the effect sizes reported with assessed statistical significance?</td>
<td>Conclusion</td>
</tr>
<tr>
<td>Q8. Are potential alternative explanations considered and discussed in the analysis?</td>
<td>Q9. Are the findings of study clearly stated and supported by the results?</td>
<td>Q10. Does the paper discuss limitations or validity?</td>
<td></td>
</tr>
</tbody>
</table>

With the question Q1, we evaluated whether or not the research objective was clearly given and backed up by industry observation or an existing theory. With questions Q2 and Q3, we wanted to see whether a study was well prepared and context information was sufficiently given. With Q4 and Q5, we assessed the sufficiency of data collection methods, instruments and measures. Besides, we wanted to see if the constructs and measures addressed the given objective. With Q6, Q7 and Q8, we evaluated if the data analysis was properly reported. For the quantitative study, we wanted to see if the effect sizes and statistical significance were reported. For the qualitative study, we wanted to see if the interpretation of qualitative evidences, such as interview quotes or observation field notes was given. We also wanted to see if other possible explanations for the result were given. With Q9 and Q10, we assessed whether study outcomes were clearly documented and the studied threat of validity was sufficiently mentioned.

We used a four points Likert scale to collect answers for quality assessment questions [68]. Each question has four possible options: an issue is not mentioned at all (score 0), the issue is rarely mentioned (score 1), the issue is adequately mentioned (score 2) and the issue is completely mentioned (score 3). To ensure the reliability of the findings of this review, we considered only the studies with the average quality score equal to or greater than 1. After this step, 45 studies were selected for data synthesis. The detailed quality assessment is given in Appendix 3.

3.5. Data analysis

Several synthesis methods, such as thematic analysis, grounded theory, meta-ethnography and case survey [47], were considered. Due to the variety of available information among studies and the previous experience of reviewers, thematic analysis was selected as a synthesis method for all RQs, as shown in Table 4. A tailor thematic synthesis was conducted, following the recommended steps proposed by Cruzes and Dybå (2011) [17]. We identified the relevant codes from the data extraction form. Since the data extraction forms already classified relevant texts in separated fields, we only need to select the relevant piece of text
and label them. After that, the codes were merged into the themes to address RQ1 (concepts and measures of each dispersion dimensions). For RQ2, to identify the relationship between concepts of dispersion dimensions and concepts of team coordination challenges, we searched “linking verbs,” such as: “impact,” ”lead,” and “cause,” which appeared in the Conclusions and Discussion section of each primary study.

As we experienced an inconsistency in findings among primary studies [55], it would be meaningful to explore studies’ characteristics that help to explain the difference. The explored factors were: research method features, such as performance measure type (perception based or direct measure), level of analysis (project, team or work item), sample size (small, medium or large), subject type and quality of study, context setting factors, such as level of technology infrastructure, dispersion range (project spread in 1, 2, or 3 continents) and global development type (inner, outsource or open source). We applied a C4.5 algorithm to build a that type of performance measure could help to reduce most inconsistency among primary studies. Therefore, we synthesized the impact of dispersion dimensions on objective and perceived team performance and software quality. We only considered geographical, temporal and work dispersion dimensions due to amount of primary studies.

To synthesize an overall impact of geographical dispersion and temporal dispersion on project outcomes in a numeric range of value, an initial option for synthesizing RQ3 and RQ4 was meta-analysis. The primary studies on geographical and temporal dispersions had consistent conceptualization on dispersion dimensions and project outcomes. However, the operationalization of dependent variables (project productivity and software quality) varied a lot. Besides, the difference in statistical analysis and unit of analysis also made the primary studies more heterogeneous. Therefore we decided to choose a tailored vote counting to summarize the study results [41, 61]. Vote-counting is performed for studies that report significance level tests, i.e. p-values [61]. Different outcomes of the hypotheses tests are categorized into positive, negative and non-significant effect. Each study will give a “vote” to support one of these relationships. While p-values reveal whether a finding is statistically significant, effective size shows practical significance, or meaningfulness [74]. A cut-off point was often used in vote counting for rejecting a hypothesis when there is no effect between two variables [61].

In this study, we only counted a valid vote from a quantitative study that provides the statistical significance of the effect size. The common statistical significance level in SE are p<=0.05 and p<= 0.01 [74]. We used a threshold of significance level at 0.05 to maximize the number of likely useful measures. A vote is only valid when the effect size is practically significant. We only counted a vote from a study with a medium or large effect size [74]. For a study that interpretation of the used effect size was not available, or the effect size was not reported, we used the interpretation of the authors to decide whether to count the study.

<table>
<thead>
<tr>
<th>RQ</th>
<th>No of studies</th>
<th>Synthesis method</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>45</td>
<td>Top-down thematic</td>
<td>A framework of dispersion dimensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>synthesis</td>
<td></td>
</tr>
<tr>
<td>RQ2</td>
<td>26</td>
<td>Bottom up thematic</td>
<td>A list of coordination problems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>synthesis</td>
<td></td>
</tr>
<tr>
<td>RQ3</td>
<td>16</td>
<td>Thematic synthesis</td>
<td>Impact of geographical and temporal dispersion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vote counting</td>
<td></td>
</tr>
<tr>
<td>RQ4</td>
<td>09</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The vote was counted by a type of relationship, i.e. geographical dispersion and team performance. The ratio of votes to the total number of studies showed the trend of the relationship and addresses the RQ3 and RQ4. We used a threshold value of 66% of outcomes to reject the hypothesis that a measure has no effect on a project’s outcome measure.

3.6. Demographics

3.6.1. Distribution by year and publish source

Figure 2 shows that selected papers were published between 2001 and 2013. The first included paper was published in 2001. The peak duration for studies about dispersion and team coordination in GSD is from 2006 to 2008 (21 papers, 47%). Many conferences and workshops on GSD team coordination started in this timeframe, such as International Conference on Global Software Engineering (ICGSE 2006) and International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2008).

The selected papers come from 25 journals, 19 conferences and one workshop. They are from three main fields: SE, IS and organizational science. The publication channels contributing more than two studies are, International Conference on Software Engineering (7 papers, 15%), International Conference on Global Software Engineering (4 papers, 9%), MIS Quarterly (3 papers, 7%) and Organization Science (3 papers, 7%).

Authors contributing more than two studies are: Jim Herbsleb (7 papers, 15%), J. Alberto Espinosa (7 papers, 15%), Marcelo Cataldo (6 papers, 13%), Nachi Nagappan (4 papers, 9%), Jonathon N. Cummings (3 papers, 7%), and Darja Šmite (3 papers, 7%). This author list should be viewed as a group of authors that influenced the results of this systematic review, rather than a group of people influencing research on GSD.

3.6.2. Distribution by data collection and analysis methods

The most common study design is case study, with predetermined study protocol and data collected by multiple methods. The most common data source is data archive, with the main data source coming from source version control, collaboration tracking systems and mailing list (16 papers). It is followed by interviews (11 papers), survey (9 papers) and multiple data sources (7 papers). Interestingly, there is only one study based on data from observation and one experiment study. The distribution of data source and data analysis is given in Table 5.

![Figure 2: Distribution of publications by year](image)
Table 5: Data collection and analysis methods

<table>
<thead>
<tr>
<th>Data source</th>
<th>Data analysis</th>
<th>No.</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data archive</td>
<td>Regression analysis</td>
<td>13</td>
<td>S38, S39, S22, S28, S17, S03, S10, S12, S31, S30, S36, S27, S26</td>
</tr>
<tr>
<td>Mixed sources</td>
<td></td>
<td>3</td>
<td>S33, S19, S04</td>
</tr>
<tr>
<td>Survey</td>
<td></td>
<td>4</td>
<td>S21, S13, S23, S01</td>
</tr>
<tr>
<td>Experiment</td>
<td></td>
<td>1</td>
<td>S16</td>
</tr>
<tr>
<td>Interview</td>
<td></td>
<td>1</td>
<td>S32</td>
</tr>
<tr>
<td>Data archive</td>
<td>Correlation analysis</td>
<td>2</td>
<td>S45, S02</td>
</tr>
<tr>
<td>Mixed sources</td>
<td></td>
<td>2</td>
<td>S07, S25</td>
</tr>
<tr>
<td>Interview</td>
<td>Qualitative analysis</td>
<td>10</td>
<td>S14, S43, S24, S37, S5, S15, S35, S11, S20, S18</td>
</tr>
<tr>
<td>Survey</td>
<td></td>
<td>4</td>
<td>S08, S26, S42, S40</td>
</tr>
<tr>
<td>Mixed sources</td>
<td></td>
<td>3</td>
<td>S09, S06, S41</td>
</tr>
<tr>
<td>Data archive</td>
<td></td>
<td>1</td>
<td>S05</td>
</tr>
<tr>
<td>Observation</td>
<td></td>
<td>1</td>
<td>S44</td>
</tr>
</tbody>
</table>

Qualitative studies mainly focus on identifying the dispersion dimensions (RQ1), and coordination challenges (RQ2) (19 papers). Among these studies, there are 10 studies using data from interviews, four studies using survey data and three studies using multiple data sources. Quantitative studies focus on validating relationships between dispersion dimensions and project outcomes (RQ3, RQ4) using some kind of regression analysis, such as linear regression, logistic regression, hierarchical regression and negative binomial regression (26 papers). Some studies performed a data transformation and other kinds of data pre-process before applying regression models. Among quantitative studies, there are 4 studies which do not perform regression analysis (S02, S07, S25, S45).

3.6.3. Distribution by dispersion setting
Figure 3 shows that there are 30 studies investigating the setting within a single organization (intra-organization). Six studies report the results from both inter and intra organizational relationships. These studies either have multiple datasets from different dispersion context settings or perform a survey with mixed companies. There are only four studies investigating inter-organizational software projects. Three studies are performed in laboratory or university settings. Only two studies occurred in open source project context. Consequently, the result is expected to report less coordination problems in regards to inter-organizational context, such as organizational and work dispersion dimensions.

4. Results

4.1. How are dispersion dimensions defined and measured in GSD studies?
Each dispersion dimension is described by its conceptualization and operationalization, as shown in Table 6. At the conceptual level, geographical dispersion and temporal dispersion are consistently understood. In the operational level, geographical dispersion is measured by physical distance or extents of physical separation perception of team members on the distance and direct measures of the distance. Temporal dispersion is measured by working time difference the extent of temporal dispersion across locations.
Interestingly, many studies use dichotomous variables to compare and contrast performance and the quality of collocated tasks and distributed ones. This measure is a proxy for all other dispersion dimensions, so it would not be sufficient to determine anything about the impact of any specific dimension of global context setting.

Conceptually, work dispersion is referred as (1) variations in engineering process, (2) differences in task and expertise distribution across locations, (3) differences in work environment, tools and (4) development practices. Some metrics, such as the portion of efforts distributed among locations and difference in CMMi level, are quantified for the first two aspects of work dispersion. In our search scope, we found no study focused on quantifying aspects of work environment, tools and practices and exploring its relationship to project outcomes.

Cultural dispersion is commonly referred to as cultural, linguistic, background differences among team members. There is one study quantifying cultural dispersion using Hofstede culture factors, such as power distance, individualism, masculinity, uncertainty avoidance, and long-term orientation (S24). In our primary studies, we only found one study associated with an objective measure of cultural dispersion and project outcomes (S32).

Organizational dispersions are also conceptualized in various ways. Organizational dispersion occurs when (1) a team consists of members from different organizations or independent organizational units; (2) there are differences in objective and development strategy among organizational units; (3) there are differences in organizational structure across projects and (4) existence of different responsibility sharing and team identity.

Table 6: Conceptualization and Operationalization of Global dispersion dimensions

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Conceptual level</th>
<th>Operational level</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical (38 studies)</td>
<td>Team including members locate in different physical places</td>
<td>Perception: Ease of travel, relocation among sites</td>
<td>S14, S11, S33, S42, S44, S15, S08, S07, S23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S27, S22, S04, S18, S28, S31, S19a, S19b, S17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dichotomous variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Based on physical distance among team members (in mile/km)</td>
<td>S03, S32, S29, S30, S39, S25</td>
</tr>
<tr>
<td>Team contains members from different building/campus/locality/country/continent</td>
<td>S10, S21, S13, S02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of locations</td>
<td>S34, S45, S30, S39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrated in study context</td>
<td>S26, S35, S05, S20, S40, S41, S43, S37, S09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal dispersion (19 studies)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>There are difference in working time among distributed team members</td>
<td>Perception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Based on team zone difference by hours</td>
<td>S03, S32, S13, S15, S29, S30, S39, S01, S25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in start working time</td>
<td>S12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ranked variable about temporal discontinuity</td>
<td>S21, S13, S16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work dispersion (12 studies)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in engineering process</td>
<td>Perception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process maturity level</td>
<td>S29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in task and expertise distribution</td>
<td>Ratio of efforts between two different locations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S38, S30, S39, S25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in tool adoption and practices</td>
<td>Perception</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S44, S24, S23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cultural dispersion (11 studies)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in national and cultural background and customs, religion, politics, language</td>
<td>Perception, i.e Hofstede dimensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S11, S42, S06, S24, S44, S15, S01, S37, S35, S23, S24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in communicating language</td>
<td>S32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational dispersion (9 studies)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in organizational bond</td>
<td>Organizational ownership</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S03, S43, S23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Role, title, position in projects</td>
<td>S32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team commitment</td>
<td>S14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in organizational objective and strategy</td>
<td>Perception</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in organizational structure</td>
<td>Number of organization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S03, S14, S26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational cohesion of contributors</td>
<td>S36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference in team identity</td>
<td>Perception</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 4 shows the extent to which dispersion dimensions (Geographical (G), Temporal (T), Cultural (C), Work and process (W) and Organizational (O)) are exposed in selected primary studies. Most of the papers investigated geographical dispersion as a general phenomenon or were briefly described in study context (17 studies). There is only one study considering all five dispersion dimensions. Therefore, it appears that global dispersion dimensions are not fully considered as project outcome influencing factors in GSD studies. Specifically, there is a relatively small amount of studies associating measures of organizational and cultural dispersions and project outcomes.

4.2. RQ2: What are the different coordination challenges that dispersion dimensions present to GSD project outcomes?

Table 7 describes 15 concepts of coordination problems related to the global dispersion dimensions. As a common perception, geographical dispersion leads to a decrease of communication frequency (S02, S03, S04, S12, S15, S22, S28, S29, S30, S32, S35), large communication networks (S02, S03, S04, S13, S17, S26, S24, S18, S28, S06, S14), and difficulty in finding relevant stakeholders (S01, S02, S03, S04, S11, S15, S16, S26). Temporal dispersion leads to time lag in communicating tasks (S06, S07, S16, S21, S28), difficulty in scheduling meetings (S32, S33) and limited ability to communicate synchronously (S12, S19, S32). Cultural dispersion’s largest issue is misinterpretation during all types of development activities: communication, requirement elicitation, development, maintenance, and testing, is the most mentioned (S15, S21, S19, S20, S37). Besides, mismatches in language capability influence the choice and quality of communication means. This observation is in line with previous reviews [18, 35].
Table 7: Coordination problems in dispersed setting

<table>
<thead>
<tr>
<th>Coordination Issues</th>
<th>Dispersions G</th>
<th>T</th>
<th>C</th>
<th>O</th>
<th>W</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrease of communication frequency</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>Large communication network</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Difficulty in finding relevant expertise/ Coordination requirement</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Lack of trust</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Lack of team identity/ team awareness</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Delay in communication and coordination</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Difficulty in organizing task</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Limited choice of communication mean</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Developing informal communication</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Misinterpretation of tasks</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>Extra coordination due to mismatches in goals, perceived value</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Extra coordination due to mismatches in organizational structure</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Extra coordination due to different local management policy</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Complicated communication and coordination paths</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Difficulty in identifying role and responsibility</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>37</td>
<td>16</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>82</td>
</tr>
</tbody>
</table>

The main coordination issue of **organizational dispersion** is the coordination overload due to mismatches in goals, perceived value and misinterpretation of tasks. Especially, the client-vendor communication barriers negatively moderate the relationship between client coordination (S06), such as influencing the understanding of requirements and knowledge sharing between the client and vendor teams (S44). Besides, differences in organizational affiliations can inhibit a group's ability to develop a shared sense of identity (S29, S30). Many organizations involved in distributed projects are not ready to change their internal structure and processes with new partners (S21). The mismatch of team structure among different sites hinders site-cross communication and the ability to satisfy coordination requirements (S26).

The main coordination issue of **work dispersion** is the increment of complexity and cost of coordination, make it harder to divide tasks and assign responsibility across sites than in a homogeneous environment (S07, S22). The mismatch in role structure made it difficult to identify team roles and responsibilities in distributed context, especially the differences in work practices and (S07). The distribution of project phases among sites leads to problematic overall joint management, problematic responsibility share and extra management needed at every location (S21). The mismatch in development and collaboration infrastructure among sites hinders frequency and quality of site-across communication (S22). The mismatch in uses of terminologies introduces difficulty in shared understanding, and leads to coordination time delays and possible reworks (S22). The mismatches in management approaches create conflict and rework in cross-site coordination (S23).

**4.3 RQ3: How do dispersion dimensions affect team performance in a GSD project?**
Tables 8-11 describe the findings from the primary studies about the impact of geographical and temporal dispersion dimensions on project outcomes. Each study might have a different Unit of Analysis (UoA), measure of independent factors and dependent factors. Measure of project outcomes can be either objective or perceived type. Team performance objective measures are (1) task resolution time, (2) KLOC/person hour, (3) time between release and (4) project duration, while the subjective measure is the perception of project managers on project schedule, budget and quality. Software quality studies use objective measure, such as number of defects, and failure proneness and perceived measure, such as perceived quality of source code and the conformance of product to client requirements.

4.3.1. Geographical Dispersion and Team performance

As shown in Table 8, 71% of studies report a negative impact of geographical dispersion on objective team performance. Herbsleb et al. (2003) found that the mean resolution time of distributed work items is two and half times longer than that of single-site work times (S19a, S19b). The main difference between collocated tasks and distributed tasks is that distributed communication networks require more people to participate, thereby introducing a delay. Cataldo et al. (2006) investigated how the satisfaction of coordination needs influence task resolution time in GSD projects (S27). Controlling task characteristics like change size, priority and release, the authors found that when developers are geographically distributed, the amount of time required to resolve modification requests is likely to increase. Espinosa et al. (2007) found that geographic dispersion and team size have a negative impact on team performance, and this impact is mediated by team familiarity (S17). Without team familiarity, geographically dispersed team members need to bridge the geographical distance to coordinate their work in some other way, making it more difficult to work together because of a lack of presence awareness, frequent communication, and contextual reference.

T. Nguyen et al. (2008) performed a correlation analysis on the number of locations, task response time and resolution time in Jazz teams (S45). Similar to findings from S19, the mean resolution time increases as the number of sites increases from one to four sites. However, when a task is distributed in five locations, the mean resolution time is reduced. The statistically small correlation coefficient between the number of sites and resolution time led to a conclusion that distributed development does not introduce a significant amount of delay compared to same-site development. The authors claimed that a construct of community of effective collaboration practices supported by advanced technology helps in solving the problem of geographical dispersion and other dimensions.

In contrast to previous findings, Ramasubbu et al. (2011) investigated 362 GSD projects from four organizations and showed that geographical distance leads to higher productivity (S39). The authors found that a unit increase in geographical dispersion leads to an increase of 1.16 KLOC per person. One possible explanation for this striking result is that while collaboration between a pair of developers is delayed, at the project level, firms gain the benefits of being distributed. An alternative explanation is that the relationship headquarters and development centers are mature enough to mitigate the negative impact of geographical dispersion. Besides, in three organizations, the involved development centers have a high level of process maturity, such as ISO 9001:2000 and CMMI 5, which facilitates a formulation of global processes and practices across sites.
Table 8: Geographical dispersion and team performance

<table>
<thead>
<tr>
<th>Studies</th>
<th>UoA</th>
<th>Independent factor</th>
<th>Dependent factor</th>
<th>Outcome type</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Herbleb et al. 2003 (S19a)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Task resolution time</td>
<td>objective</td>
<td>-</td>
</tr>
<tr>
<td>Herbleb et al. 2003 (S19b)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Task resolution time</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cataldo et al. 2006 (S27)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Task resolution time</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Espinosa et al. 2007 (S17)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Task resolution time</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cataldo et al. 2008 (S28)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Task resolution time</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>T. Nguyen et al. 2008 (S45)</td>
<td>Task</td>
<td>No of site</td>
<td>Task resolution time</td>
<td>-</td>
<td>O</td>
</tr>
<tr>
<td>Ramasubbu et al. 2011 (S39)</td>
<td>Project</td>
<td>Physical distance No of site</td>
<td>KLOC/ person hour</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Cramton et al. 2005 (S04)</td>
<td>Team</td>
<td>Team with distributed members</td>
<td>Judgment on team performance</td>
<td>perceived</td>
<td>-</td>
</tr>
<tr>
<td>Hoegl et al. 2007 (S33)</td>
<td>Team</td>
<td>Team proximity</td>
<td>Effectiveness, efficiency</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Damian et al. 2007 (S07)</td>
<td>Task</td>
<td>Distributed vs. Collocate</td>
<td>Communication frequency</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cumming et al. 2007 (S21)</td>
<td>Team</td>
<td>Spatial/ Temporal boundaries</td>
<td>Coordination delay</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Chudoba et al. 2005 (S23)</td>
<td>Team</td>
<td>Team distribution</td>
<td>Team performance</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Espinosa et al. 2011 (S13)</td>
<td>Project</td>
<td>Spatial boundary</td>
<td>Team performance</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Table 8 also shows that 66% of studies reported a negative impact of geographical dispersion on perceived team performance. Cramton et al. (2005) surveyed 218 team members from 39 dispersed teams and showed that teams with geographically dispersed members reported significantly less effective performance than collocated teams (S04). Cummings et al. (2007) surveyed 5,986 dispersed team members in a large cooperation and found that spatial boundaries is positively associated with coordination delays for pairs of global team members, which lead to lower perceived team performance (S21). Consistent with S04, Damian et al. (2007) found a significant decline of communication frequency over geographical distance (S07). Hoegl et al. (2007) conducted structured interviews on 145 team members and found that proximity of team members mediate the influence of quality of teamwork on managers’ evaluation of team performance (S33). As the team member proximity decreases—that is, as team members are more geographically dispersed—the influence of teamwork quality on team performance increases.

In two other surveys, no statistical relationship between geographical dispersion and perceived team performance was found. Espinosa et al. (2011) surveyed 123 projects in a
large firm and after controlling temporal dispersion variables; there is no relationship between geographical dispersion and coordination problems (S13). The authors conjectured that with substantial experience working globally, the firm had learned to work across distance and managed to bridge spatial separation effectively with collaboration technology. Chudoba et al. (2005) surveyed 1,269 team members in Intel about the impact of team virtuality and found no relationship between team distribution and performance (S23). One possible explanation is that Intel has adapted to remote work environments by developing continuities such as sensitivity to cultural and organizational differences caused by distance, and strategies to embrace these differences.

### 4.3.2 Temporal dispersion and team performance

As shown in Table 9, 100% of studies reported a positive relationship, revealing a trend that a team with large temporal dispersion is more productive than a team with small temporal dispersion. (S16, S12, S01, S39). Espinosa et al. (2007) performed an experiment on 42 teams working in four temporal settings (full overlap, 1/3, 2/3 and no overlap) (S16). An interesting finding is that compared to full overlap, there was no development speed difference in no overlap condition, but speed decreased significantly with very small amounts of time separation (i.e., 2/3 overlap). The authors implied that teams can work efficiently and uninterrupted when they have no overlap and can articulate and interpret instructions clearly. However, when teams need to constantly change their interaction mode from synchronous to asynchronous, they need to make mental adjustments in their work styles and coordination tactics, thus reducing their efficiency.

Colazo (2012) studied 100 open source projects and found that inter-release time and the number of lines written per developer increased when temporal variation among team members increased (S12). The given reason is that mechanistic coordination, which is effective for temporal dispersion is beneficial for the team as a whole, rather than for an individual developer. For a pair of developers, a large time zone difference between them hinders the synchronous team collaboration and hence reduces the team performance. But for a project with many teams of developers, large time zone differences boost project productivity if the task is appropriately divided into different time zone.

**Table 9: Temporal dispersion and objective performance**

<table>
<thead>
<tr>
<th>Studies</th>
<th>UoA</th>
<th>Independent factor</th>
<th>Dependent factor</th>
<th>Outcome type</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espinosa et al. 2007 (S16)</td>
<td>Team</td>
<td>Degree of time zone overlap</td>
<td>Task resolution time</td>
<td>objective</td>
<td>+</td>
</tr>
<tr>
<td>Colazo et al. 2010 (S12)</td>
<td>Project</td>
<td>No. of hour</td>
<td>Inter release time</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Gopal et al. 2011 (S01)</td>
<td>Project</td>
<td>No. of hour</td>
<td>Project duration</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Ramasubbu et al. 2011 (S39)</td>
<td>Project</td>
<td>No. of hour</td>
<td>KLOC/ person hour</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Cummings et al. 2007 (S21)</td>
<td>Team</td>
<td>No. of hour</td>
<td>Project on schedule, on budget and meeting requirements</td>
<td>perceived</td>
<td>-</td>
</tr>
<tr>
<td>Espinosa et al. 2011 (S13)</td>
<td>Project</td>
<td>No. of hour</td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Gopal et al. (2011) surveyed 83 projects in 9 firms and found that temporal dispersion is positively associated with development speed (S01). The study stated that “follow-the-sun” practices allow managers to leverage 24 hour work-day in making more effective use of calendar time on a project. As time separation is larger, i.e., India and the US versus India and Singapore, the team has more advantages in productivity. Ramasubbu et al. (2011) investigated 362 projects and found a positive relationship between time zone differences and the number of KLOC per person hour (S39).

Espinosa et al. (2011) performed a survey on 123 projects in a large firm and found that inferior performance was due primarily to coordination problems created by a large time zone span (S13). When team coordination is taken care of with effective task organization (i.e., role assignments, division of labor and responsibilities, work rules and routines) the negative effects of maximum time zone span on team performance can be mitigated to some extent.

4.3.3. Work dispersion and Team performance

In regards to studies of work dispersion, it seems that inappropriate dispersion of task and variety of practices has a negative impact on team performance. Ramasubbu et al. (2008, 2011) studied 362 projects and showed that increasing personnel imbalances across the sites by one standard deviation increases productivity by 2.14 KLOC per person hour (S38, S39). According to the authors, a ten-person project team configured as 8-2 between New York City and Bangalore is more productive than 6-4 and 5-5 configurations (S39). In other study, Ramasubbu et al. (2008) found that work dispersion change leads to an increase of productivity of 0.35 FPs/ person hour (S38). The authors reasoned that information asymmetry between the remote teams hinders coordination and task orchestration and ambiguous authority due to lack of complete control over the processes at both the remote sites. Chudoba et al. (2005) investigated team virtuality in Intel context and found that a variety of practices is negatively associated with team performance (S18). The symmetry of teams across sites can exaggerate the mismatches in work and management practices among sites and hence, leads to inferior team performance.

4.4 RQ4: How do dispersion dimensions affect software quality in a GSD project?

4.4.1. Geographical dispersion and Software quality

Table 10 shows that 77% of studies reported a negative impact of geographical dispersion on software quality. Interestingly, there is one study that did not find an influence of geographical distance on software quality. Bird et al. (2009) investigated Microsoft Windows Vista project and found that there is no significantly correlation between different degrees of geographical distribution and software quality (S02). This leads to the surprising conclusion that teams that were distributed wrote code that had virtually the same number of post-release failures as those that were collocated.

In open source projects, Bird et al. (2012) performed a study of Firefox and Eclipse and found that all geographical distribution measures increase the number of failures at the file level (S03a, S03b). In the Firefox case, the authors showed that in general, distribution measures are more significant prior to release than after. Higher values of spatial and worldwide distribution were associated with more defects in both prior to both releases (S03a). In the Eclipse case, there was also a mixed finding when all measures of geographic and organizational distribution increased failures, but the effects are not consistent across releases.
A similar result was observed in Kocaguneli et al. (2013) (S10). The author claimed that the size of the effect between colocated and distributed development was too small to be practically significant.

In contrast, all of the other studies found a negative impact of being geographically dispersed on software quality. Cataldo et al. (2009) investigated 562 components from GSD projects and found that an increase in distances among locations leads to an increase of the number of defects (S29). A similar finding was observed at the project level (S30). In another study, Cataldo et al. (2011) found that when developers that worked on the feature were geographically distributed, the likelihood of integration failures is almost 14 times higher than when all developers are in the same location (S31). Audris et al. (2010) studied distributed files in a large telecommunication firm and found that the number of geographic locations and the remoteness of the developer’s mentor increased fault proneness (S26). The work from multiple sites increases fault-proneness by 15% and having a remote mentor by 68%. Ramasubbu et al. (2011) studied 362 projects from four different firms and found that an increase in a unit of geographical dispersion leads to an increase in 1.26 delivered defects per KLOC (S39).

### 4.4.2. Temporal dispersion and Software quality

Table 11 shows that four studies reported a negative impact of temporal dispersion on software quality (66%), one study reported a neutral impact and one study found a positive influence of temporal distance on software quality. In an experimental setting, Espinosa et al. (2007) showed that the team with more time separation (i.e., 1/3 and no overlap) had significantly lower levels of accuracy, but a small amount of time separation (i.e., 2/3 overlap) had no effect on accuracy (S16a). The experimental subject also perceived that inferior quality occurred when work is spread out through temporal distance (S16b).
### Table 11: Temporal dispersion vs. software quality

<table>
<thead>
<tr>
<th>Studies</th>
<th>Ref.</th>
<th>UoA</th>
<th>Independent factor</th>
<th>Dependent factors</th>
<th>Outcome type</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espinosa et al. 2007</td>
<td>S16a</td>
<td>Task</td>
<td>Degree of distribution</td>
<td>No. of correct task</td>
<td>Objective</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>S16b</td>
<td></td>
<td></td>
<td>Perceived task quality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gopal et al. 2011</td>
<td>S01</td>
<td>Project</td>
<td>No. of hours</td>
<td>Perception on software product conforms to client requirements</td>
<td>Perceived</td>
<td>-</td>
</tr>
<tr>
<td>Cataldo et al. 2009</td>
<td>S29</td>
<td>Task</td>
<td>Physical distance</td>
<td>No. of defects</td>
<td>Objective</td>
<td>-</td>
</tr>
<tr>
<td>Cataldo et al. 2010</td>
<td>S30</td>
<td>Project</td>
<td>Physical distance</td>
<td>No. of defects</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Colazo et al. 2010</td>
<td>S12</td>
<td>Project</td>
<td>No. of hours</td>
<td>No. of bugs</td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Gopal et al. (2011) also found that temporal dispersion negatively influences software quality in the outsourcing context (S01). Both studies showed that geographical dispersion is less important than temporal dispersion on affecting software quality. Perhaps, teams with some work time overlap can benefit from advanced collaboration technology in which they could resolve miscommunication issues during coding and bug fixing process. Mechanistic coordination is not a preventive approach for software quality as they can facilitate the distributed bug fixing process. Besides, the strategy of coordination in specific project context may be focused on team productivity and relaxed on software quality (S01).

Cataldo et al. (2010) found that components that are developed by a team with a higher level of temporal dispersion exhibited lower levels of quality (S29). However, the authors did not find a significant relationship between temporal dispersion and software quality at the project level (S30). One possible explanation could be from the nature of the development tasks, as they would not benefit much from synchronous communication, one process underlying the impact of temporal distribution.

In open source context, Colazo et al. (2011) investigated 100 projects and found that the higher temporal dispersion is associated with fewer defects and subsequently with a higher quality of coding (S12). Besides, the positive effect of temporal boundary on quality is greater in more complex tasks. The given reason is that temporal dispersion allows developers to take time to focus on the problems at hand through comprehension of the received information and the responses that should follow, as well as allowing members to consult other resources for better problem solving effectiveness. However, open source projects (at least the selected projects in the study) do not have similar characteristics of proprietary projects, such as pressure of deadline, formal development processes and tight management and control [38].

### 4.4.3. Work dispersion and Software quality

Regards to studies of work dispersion, the mismatches in work and team dispersion lead to lower software quality. Ramasubbu et al. (2011) found that increasing personnel imbalances across the sites by one standard deviation increases defects per KLOC by 2.64 KLOC per person hour (S39). Alternatively, a ten-person project team configured as 8-2 between New
York City and Bangalore is more defect prone than 6-4 and 5-5 teams are (S39). Uneven distribution of work items, i.e. modification request, across development locations negatively impact software quality. Cataldo et al. (2006, 2008) showed that a unit increase in the dispersion measures increases the likelihood of defect as much as an unit decrease in the average process maturity in the project (S27, S28). The argument is that balanced distributions of engineers may allow individuals at each location to better manage misunderstandings across location because they have a “minimal” critical mass of developers that might be able to contribute knowledge about various parts of the system to resolve the misunderstanding locally.

5. Discussion
5.1. Summary findings

In response to RQ1, this review reveals that GSD studies have not fully considered the impact of different global dispersion dimensions on team coordination and project outcomes. Organizational dispersion has received the least attention from GSD researchers. While geographical dispersion and temporal dimensions are consistently conceptualized and quantified, cultural, work and organizational dispersion dimensions are exposed in different ways.

In response to RQ2, dispersion dimensions are related to a distinct set of coordination challenges. In line with previous studies [18, 35, 59], geographical, temporal and cultural dispersion is related to the decrease of communication frequency, delay in communication response and misinterpretation of tasks. Our novel finding is that work and organizational dispersions are associated with coordination issues at the management level, rather than at the technical level.

Table 12: Summary of findings

<table>
<thead>
<tr>
<th>RQs</th>
<th>Propositions</th>
<th>Studies</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>Organizational dispersion is the least common dimension being explored in the relationship with project outcomes</td>
<td>Found in 45 studies</td>
<td>Section 4.1</td>
</tr>
<tr>
<td></td>
<td>The impact of different dispersion dimensions on project outcomes were not fully discovered in GSD literatures</td>
<td>Found in 45 studies</td>
<td>Section 4.1</td>
</tr>
<tr>
<td>RQ2</td>
<td>Dispersion dimensions are related to distinct sets of coordination challenges</td>
<td>Found in 45 studies</td>
<td>Section 4.2</td>
</tr>
<tr>
<td>RQ3</td>
<td>A distributed task tends to take longer time to communicate and resolve than a collocated task does</td>
<td>Supported by 5/7 studies</td>
<td>Section 4.3.1</td>
</tr>
<tr>
<td></td>
<td>Geographical dispersion tends to have a non-positive impact on the perception of team members on their performance</td>
<td>Supported by 4/6 studies</td>
<td>Section 4.3.1</td>
</tr>
<tr>
<td></td>
<td>Temporal dispersion tends to have a positive impact on objective team performance, while it has a negative impact on perceived team performance</td>
<td>Supported by 6/6 studies</td>
<td>Section 4.3.2</td>
</tr>
<tr>
<td>RQ4</td>
<td>Geographical dispersion tends to have a negative impact on software quality, at both file and project level</td>
<td>Supported by 8/9 studies</td>
<td>Section 4.4.1</td>
</tr>
<tr>
<td></td>
<td>Temporal dispersion tends to have a negative impact on software quality, at both file and project level</td>
<td>Supported by 4/6 studies</td>
<td>Section 4.4.2</td>
</tr>
</tbody>
</table>
In response to RQ3, this review confirms the common perception that geographical dispersion has a negative impact on actual team performance and also the perception of stakeholders. It seems that team familiarity and adoption of collaboration technology could help to reduce the negative impact of geographical dispersion. The primary studies also revealed a trend that temporal dispersion has a positive impact on objective team performance, while it has a negative impact on perceived team performance. The impact of global dispersion might be different in an organization with a long experience of working from distances.

In response to RQ4, geographical dispersion shows a tendency of a negative impact on software quality. Temporal dispersion also tends to have a negative impact on software quality. It is interesting that a study on open source projects shows a positive impact of temporal dispersion on software quality. The summary of findings as propositions is shown in Table 12.

5.2. Implication for future research

The review reveals a limited number of empirical studies on the relationship between organizational dispersion and team coordination, which is in line with findings from a previous review [18]. Curtis et al. noted that coordination breakdowns were likely to occur at organizational boundaries, but coordination across these boundaries was often extremely important to the success of the project [5]. A future study in the context of partnership in a large-scale government project [76], supply chain in automobile industry [67], collaboration among autonomous branches of an international organization [53], cooperation between acquiring company and acquired company [54], and open software ecosystem [16], would bring more variety in GSD literature.

In this review, we only found one study that investigated the influence of work process diversity on team coordination and communication (S08). Given the important roles of collaboration technology and practice in GSD [4, 24, 45, 79], the adoption of shared artifact and process mechanism to manage this type of dependency is still an open area for research. Future studies can explore how mismatches in development processes and practices among different locations impact team coordination and project outcome.

Our review also found primary studies that suggested a learning process in dealing with coordination challenges in GSD (S28, S15, S17, S41). The emergent of coordination challenges and effectiveness of coordination mechanism is hard to be observed in cross sectional studies. We only found two longitudinal studies, which address this research area (S41, S43). Future studies could address how team coordination and communication work over time.

Our review revealed only two studies in the context of open source projects. Open source projects can represent an extreme case of cultural dispersion and organizational dispersion is also large in open source ecosystem [52]. Future studies on the influence of dispersion dimensions on open source project performance and quality, a comparison and contrast findings with studies in closed source projects can bring practical lessons in GSD literature.

5.3. Implication for practitioners

Our synthesis of 45 relevant studies showed a trend of negative impacts of different types of
dispersion on team performance and software quality. In some cases, the impact is significant and leads to the failure of the project. Therefore, managers should not discount this impact of global context setting while estimating the potential cost-benefits of their globally dispersing software development strategy. Potential savings from dispersing work to offshore centers have to be weighed against the possible loss of overall development productivity and software quality.

GSD managers should be aware of the influence of dispersion dimension at different organizational levels. At the individual level, a lack of face-to-face interaction and working in different time zones directly and negatively affects a developer’s work. At team and project levels, the negative influences of these dispersion dimensions might be underestimated in considering different goals and priorities. This issue must be taken into account when aligning the objective of individuals and teams with organizational goals.

Decisions on what coordination mechanism to use should depend on the current dispersion context setting, the current team coordination technology and practices and prioritized type of interdependencies. Although managers cannot entirely surmount the challenges associated with bridging these boundaries, they can mitigate them by a combination of different coordination mechanisms. Problems with geographical dispersion and small temporal dispersion could be mitigated by selective and structured communication. Significant temporal dispersion where synchronous communication is not possible, task schedule and united work process is beneficial for team coordination. Cultural dispersion might be reduced by formal communication and early standardization activities, such as team training and site visits. Work process and organizational dispersion should be managed by situational and balanced coordination mechanisms. Last but not least, the same coordination mechanism may be applicable at an individual level, but not suitable at a team level. Evidence from empirical studies showed that communication and shared artifacts should be used in pairs as needed and a defined process should be adopted at the team and organizational levels.

Our primary studies are consistent on the negative impact of geographical and temporal dispersion on software quality. Therefore, with a quality oriented products, distributing teams across geographical locations and time zone might not be an effective option. Traditional quality assurance approaches in collocated teams might need to consider the differences in time zones, lack of face-to-face communication and variety in processes and practices among locations.

6. Threats on validity

6.1. Threats to the search of publications

One common threat to systematic literature reviews is not to discover all of the relevant studies. The reason not to cover all seed studies may be that the search ranges are in multiple disciplines, such as SE, IS and organization and management science. To reduce this threat, we adopted a search strategy to retrieve as many documents as possible related to the research topic. By extending the search scope, we achieved 11,222 unduplicated documents from the search engines, which reflect a very low search precision (0.33%). The initial search string had 87.5% coverage and after refinement, we achieved 92.75% coverage. Besides, the additional search by using Google scholar search engine and forward, backward scanning through the relevant studies helped to further decrease the probability of missing relevant
studies. We believe that the group of missing papers is small enough to not influence the findings from our review. Another threat is publication bias, where papers reporting positive results tend to be published. We reduce this threat by searching for gray literature.

6.2. Threats to selection of publications

A relevant study may be misclassified into a removal group during the selection process and vice versa. To reduce the bias in selection of papers, we defined a review protocol with clear inclusion and exclusion criteria for each selection phases. In the first selection round, as large amount of irrelevant studies can be removed by screening titles and abstracts, the task was performed by one author without considering this threat. In the second selection round, the filtering of 470 papers was conducted by the two first authors and constantly crosschecked. We also adopted a well-prepared quality assessment checklist to judge paper quality. Any in decisions on selecting or judging paper quality was discussed until we reached consensus. Fleis’s kappa for agreement on inclusion in the review was 0.84 for the two reviewers [36]. The kappa value between 0.81 and 1.00 represents an almost perfect agreement, suggesting a very good agreement among the authors. In case that a decision could not be made by two reviewers, we involved the third reviewer in the discussion.

6.3. Threats to extraction and synthesis of publication

Primary studies use different research design, data collection and analysis methods, which leads to a threat of validity of synthesis process. To reduce the variety in primary studies, we used an extraction form, which is driven by research questions. The form had been initiated by studies in the initial pool and validated during the trial search process. Only studies that addressed at least 2 RQs were included. So we had 45 studies (100%) for RQ1, 26 studies (58%) for RQ2, 16 studies (36%) for RQ3 and 9 studies (20%) for RQ4.

During the synthesis process, we needed to consider whether to count studies using different measures and different level of analysis in the same dataset. We decided to count studies by the unit of analysis type, e.g., at work item, team or project level and by type of measures, such as perceived and objective performance. These two factors also helped to cluster primary studies into cohesive groups. Other primary study design factors, such as research design type, size of data set, data collection methods, global arrangement, number of locations and team size were also considered, but they were not helpful in explaining the divergent findings from primary studies.

6.4. Threats to external validity

As we provided conclusions about the relationship between dispersion dimensions and project outcomes, there is a risk that an actual causal relationship is not captured. As most of studies used for RQ3 and RQ4 adopted some types of multivariate regression model, the causal relationship between dispersion dimensions and project outcomes were investigated with consideration of context variables. For three studies using correlation analysis for RQ3 and RQ4, the vote from this study was counted only as a half point. However, the inherent problem of a systematic literature review, which states that the quality of synthesized results will not be better than quality of primary studies, also occurs in our study [3]. Quality assessment was been applied to reduce this threat.
6.5. Threats to generalization

The limited number of studies on each dispersion dimension (16 studies at most) did not allow us to conduct any quantitative methods, such as meta-analysis to quantify the effect size of the impact of dispersion dimensions on project outcomes. Even for vote counting, in some cases, the number of studies is too little to provide a synthesis (i.e. influence of work dispersions on project outcomes). Besides, primary studies are varied in terms of research designs, context setting and measures of dependent and independent factors. Combining such studies equals mixing apples and oranges [41]. It can be argued, however, that mixing apples and oranges is beneficial, particularly if one wants to generalize about fruit, and that the studies that are exactly the same in all respects are actually limited in generalizability [11]. In this study, we did not aim at quantifying the impact of dispersion dimensions on team performance or product quality. Instead, we concluded on impact trends for temporal and geographical dispersion dimensions. Consensus observations from these studies suggested important context factors that should be further investigated.

During the synthesis process, we also observed studies reporting contradicting outcomes for a certain relationship. By considering context factors, we found that there was no contradicting studies with the same set of independent factor measures, dependent factor measures and units of analysis. The most variation between studies lies in the statistical significant level, for instance one study found a significantly negative relationship between geographical dispersion and team performance, while another study did not find a significant relationship. By investigating the differences in dispersion settings, project context and given the explanation in the study, we came up with possible explanations for these variations.

7. Conclusions

Unlike existing literature reviews on GSD, this review synthesizes impacts of global dispersion dimensions on coordination, team performance and software quality. Our results show that global dispersion dispersions are consistently conceptualized, but are quantified in many different ways. The primary studies reveal that each of the dispersion dimensions is associated with a distinct set of coordination challenges. In regard to the impact of dispersion dimensions on team performance, we found that geographical dispersion has a negative impact on task resolution time and also on the perception of team members. However, the influence of geographical dispersion on different levels of analysis might be different. Temporal dispersion has a positive impact on objective team performance but a negative impact on perceived team performance. Geographical dispersion has a negative impact on software quality in general. Temporal dispersion has a negative impact on software quality in closed source software projects.

The findings suggest four areas for future works in GSD: (1) coordination challenges in inter-organizational projects, (2) impact of processes and practices mismatches on project outcomes, (3) evolution of coordination needs and mechanism over time and (4) impact of dispersion dimensions on open source project outcomes. We also provide four recommendations for GSD managers in regards to (1) the tradeoff between cost and benefits while dispersing tasks, (2) alignment impact of dispersion dimensions with individual and organizational objectives, (3) coordination mechanisms as situational approaches and (4) collocation of development activities of high quality demand components.
Acknowledgement

We gratefully appreciate Professor James D. Herbsleb from the Institute for Software Research, CMU for valuable discussions on the systematic review, and Associate Professor Torgeir Dingsøyr for thorough reviews of this paper.

References


178


179


of Actors in Globally Distributed Project Teams”, 7th, ICGSE, Porto Alegre, Brazil, pp. 144-148, 2012


### Appendix 1: List of selected papers

<table>
<thead>
<tr>
<th>Code</th>
<th>References</th>
</tr>
</thead>
</table>
| S17  | J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Familiarity, complexity, and team performance in geographically distributed software
<table>
<thead>
<tr>
<th>ID</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>S20</td>
<td>J. D. Herbsleb, D. J. Paulish, M. Bass, “Global software development at Siemens: experience from nine projects”, 27th ICSE, Missouri, USA, 2005, pp. 524-533</td>
</tr>
<tr>
<td>S26</td>
<td>A. Mockus, &quot;Organizational volatility and its effects on software defects&quot;, 18th SIGSOFT FSE, 2010, Santa Fe, USA, pp. 117-126</td>
</tr>
<tr>
<td>S30</td>
<td>M. Cataldo, “Sources of errors in distributed development projects: implications for collaborative tools”, CSCW, Georgia, USA, 2010, pp. 281-290</td>
</tr>
<tr>
<td>S34</td>
<td>M. V. Rini van Solingen, “The Impact of Number of Sites in a Follow the Sun setting on the Actual and Perceived Working Speed and Accuracy: A Controlled Experiment”, 5th ICGSE, Princeton, NJ, USA, 2010</td>
</tr>
</tbody>
</table>
## Appendix 2: Code extraction

<table>
<thead>
<tr>
<th>Publication feature</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>[narrative]</td>
</tr>
<tr>
<td>Author</td>
<td>[narrative]</td>
</tr>
<tr>
<td>Year</td>
<td>[narrative]</td>
</tr>
<tr>
<td>Publication source</td>
<td>Conference/ Journal/ Book name</td>
</tr>
</tbody>
</table>

### Research design

| Study type:                     | Case study/ Industry report/ Experiment/ Survey/ Action research/ Interview/ Data mining |
| Technology support level        | low/ medium/ high                                                      |
| Case study type                 | Cross sectional/ Longitudinal                                          |
| Distribution type               | Global branch/ Outsourcing/ Opensourcing/ Others                       |
| Unit of analysis                | Person/ Team/ Project                                                  |
| Sample size                     | N=?                                                               |
| Data collection method          | Interview/ Survey/ Case study/ Data archive/ …                      |
| Data analysis type              | Qualitative/ Quantitative/ Both                                       |
| Data analysis method            | Case-comparison analysis/ Grounded theory/ Correlation analysis/ Regression analysis/ … |
| Control of context factor?      | Yes/ No/ Not applicable                                              |

### Context information

| Number of site                  | 1/2/3/…/ unknown          |
| Site location                   | Cities name               |
| Technology support level        | low/ medium/ high          |

### Performance factor

| Factor explanation              | [narrative]               |
| Factor measure                  | [narrative]               |

### Dispersion factor

| Factor description              | Yes/ No/ Not mentioned   |
| Factor explanation              | [narrative]               |
| Factor measure                  | [narrative]               |

### Results

| [narrative]                      |

### Implications

| [narrative]                      |

### Threat to validity

| [narrative]                      |
## Appendix 3: Paper quality assessment

<table>
<thead>
<tr>
<th>Study</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7a</th>
<th>Q7b</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S15</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.9</td>
</tr>
<tr>
<td>S19</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2.9</td>
</tr>
<tr>
<td>S14</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2.8</td>
</tr>
<tr>
<td>S17</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>S32</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>S01</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>S05</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>n/a</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>S12</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>S13</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>S43</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>n/a</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2.7</td>
</tr>
<tr>
<td>S31</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>S35</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>S23</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2.5</td>
</tr>
<tr>
<td>S24</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2.5</td>
</tr>
<tr>
<td>S09</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>n/a</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td>S18</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2.4</td>
</tr>
<tr>
<td>S38</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2.4</td>
</tr>
<tr>
<td>S39</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2.4</td>
</tr>
<tr>
<td>S41</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2.4</td>
</tr>
<tr>
<td>S04</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>S21</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>n/a</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>S26</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2.3</td>
</tr>
<tr>
<td>S28</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>S29</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>S30</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2.3</td>
</tr>
<tr>
<td>S25</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2.2</td>
</tr>
<tr>
<td>S33</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>n/a</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2.2</td>
</tr>
<tr>
<td>S02</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2.1</td>
</tr>
<tr>
<td>S10</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2.1</td>
</tr>
<tr>
<td>S44</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2.1</td>
</tr>
<tr>
<td>S42</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>n/a</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.9</td>
</tr>
<tr>
<td>S03</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>S16</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.8</td>
</tr>
<tr>
<td>S34</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>S36</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1.8</td>
</tr>
<tr>
<td>S45</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>n/a</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>S27</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.6</td>
</tr>
<tr>
<td>S40</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>n/a</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>S08</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1.3</td>
</tr>
<tr>
<td>S20</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>n/a</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1.2</td>
</tr>
<tr>
<td>S11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>n/a</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.1</td>
</tr>
<tr>
<td>S37</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>n/a</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.1</td>
</tr>
<tr>
<td>S06</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S07</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>n/a</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S22</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>n/a</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Paper MP5 - The influence of organizational distance on technical coordination – An exploratory study

Coordination of software development teams across organizational boundary – An exploratory study

Anh Nguyen-Duc
IDI, NTNU
Trondheim, Norway
anhn@idi.ntnu.no

Daniela S. Cruzes
IDI/NTNU and SINTEF IKT
Trondheim, Norway
dcruzes@idi.ntnu.no

Abstract— Coordinating teams across geographical, temporal and cultural boundaries has been identified as a critical task to achieve the success of global software projects. Organizational boundary is another dimension of global distribution, which is a less visible but equally important factor that influences team coordination. This study investigates attributes of the organizational boundary that inhibits coordination and development activities. Besides, we explore a set of effective coordination practices to overcome organizational boundary. The data were collected from two projects involving four different software development organizations. We found that the variety on collaboration policy, team organization, engineering process, and development practices contributes to extra coordination efforts, insufficient communication, team awareness and mistrust. The study also highlights that coordination practices, such as face-to-face contact, process synchronization and shared collaborative development are compulsory but not sufficient for effective team coordination across organizational boundary.

Keywords: global software development, organizational boundary, coordination, collaboration, empirical study

1. Introduction

Global software development (GSD) promises shortening time-to-market, fastening technology innovation, increasing operational efficiency and reducing the proximity to customers [1]. However, there are a number of problems related to economical, social and technical issues that still need to be solved before the full potential of GSD can be reached. One of the main problems is that team coordination in GSD is insufficiently supported [2].

Coordination, understood as activities to manage dependencies between tasks and task holders, is essential for orchestrating a large software project. In collocated software projects, poor coordination is accounted for uncertainty and complexity of tasks [3, 4]. In GSD context, distributed software team needs to handle not only the complexity and evolution of requirements and technical tasks, but also dynamic interdependencies between tasks and task holders across multiple global boundaries. Consequently, tasks that spread through many locations take much longer time to complete than collocated tasks, and suffer from lower team productivity and software quality [5, 6].

A recent trend of research on GSD has focused much on improving socio-technical coordination [2, 7], in which project stakeholders coordinate on multiple layers of artifact dependency, task dependency and social dependency, as shown in Figure 1. Many empirical studies target to these layers, assuming that stakeholders belong to a homogeneous organizational environment. An often-neglected dimension of socio-technical coordination is the organizational boundary among stakeholders, as not all of them belong to the same
organization. Organizational boundary means that there will be differences in engineering processes, role structure, functional and strategic concerns occurred among organizational units [8]. Curtis et al. noted that coordination breakdowns were likely to occur at organizational boundaries, and coordination across these boundaries was often extremely important to the success of the project [10]. Espinosa et al. stated that organizational boundaries can affect project outcomes by reducing awareness of context and shared identity [8].

Figure 1 illustrates two types of organizational boundaries occurred at departmental level (between Team A1 and Team A2) and firm level (between Team A1 and Team A3). These boundaries introduce barriers between developer D1 and D2 and also between D1 and D3, influencing the implementation of task A1 and A2. While organizational distance between two firms is expected, we argue that there are also attributes of organizational boundary that matters for coordinating two development units within the same organization.

Our objective in this paper is to explore different attributes of organizational boundary, its consequence and remedies in the context of GSD. The first step of our study is to explore the attributes of organizational boundary and their outcomes on coordination among project team members. After identifying a list of organizational boundary attributes that cause coordination problems, we investigated the solutions that have been effectively adopted by practitioners. The two main research questions are:

1. RQ1: What are attributes of the organizational boundary that introduce challenges on team coordination in GSD?

2. RQ2: What are the coordination mechanisms that help to mitigate organizational boundary?

The extent of organizational boundary and its impact on team coordination can vary in different organizational relationships. Therefore, we examine and compare two projects with different organizational relationship, one leveraging company-internal resources in a different country (onshore outsourcing) and the other leveraging external third-party resources situated in the same country (offshore insourcing) [14].

The rest of the paper is organized as follows. In Section 2, we present an overview of related work to organizational boundaries, team coordination challenges and coordination mechanisms in GSD. In Section 3, we describe an empirical study design and context background of the cases. Results from the empirical studies are provided in Section 4. It is followed by a discussion in Section 5. Finally, Section 6 presents the conclusion with the summary of major findings.

Figure 1: Organizational perspectives on socio-technical coordination (adapted from [12])
2. Related Works

2.1. Organizational boundaries in GSD
While exploring challenges in GSD, SE researchers have found several issues with organizational boundaries, which can influence team coordination [13-20]. Divergence in terms of organization’s objectives, policies and strategies has a visible impact on developers’ work [13]. Espinosa et al. found that collaboration policy established by both side of a collaborated project affects the way work was coordinated [7].

Organizing teams across geographical and organizational boundary also plays a role in supporting team coordination. Hinds et al. found that informal hierarchical structure was associated with more smooth coordination on distributed teams [16]. Nagappan et al. investigated Vista projects and showed that measures of organizational structure have a significant relationship to software quality, and this relationship is stronger than ones of geographical or temporal factors [11].

The issue of a common process in software specification, implementation and maintenance is also nontrivial. Kommeren et al. reported experience with GSD in Philips and reported problems with lack of common process about acceptance procedures of mutual deliveries, escalation mechanisms, lack of clearly assigned responsibilities and lack of central control of integration [18]. Prikladnicki et al. showed that the software development process variation is more severe in offshore outsourcing projects than in internal offshoring projects [22].

Lastly, organizational boundary introduces not only variation in development processes, but also in development practices, such as coding and comment styles, testing approaches and adoption of development tools. The practices are likely informal and not defined by the overall process. Šmite et al. found that coordination by standard is hindered by problems with disparities in work practices and limited feedback from remote locations [19]. Moe et al. also reported challenges with acquiring trust introduced by disparities in work practices [20]. Bhat et al. showed problems with conflicting RE approaches on disagreements in tool selection and communication issues [13].

It appears that the differences in the organization’s policy, the way teams are organized, variation in development process and work practices are potential challenges for cross-team coordination. We conceptualize these attributes as Collaboration Policy, Team Organization, Engineering Process, and Development Practices and use this as an initial checklist to explore the organizational boundaries.

2.2. Team coordination challenges in GSD
In the scope of this study, we use the term “collaboration” to refer to the act of working with other to create something together in general. The term “coordination” refers to “activities required to maintain consistency within a work product or to manage dependencies within the workflow” [21]. Several types of dependencies have been identified in the context of Information System (IS) and Software Engineering (SE), such as [21, 22]: technical dependency (i.e., needed to integrate software parts seamlessly), temporal dependency (i.e., necessary to synchronize activities and adhere to project schedules), software process dependency (i.e., required to adhere to the established software processes and act as agreed with other team members), and resource dependency (i.e., a team needed business domain
knowledge from the other team to accomplish a task). Insufficient management of these dependencies causes coordination problems. The challenges identified in previous literature are [3, 9, 20, 23-28]:

1. Large and complex dependency network: many stakeholders from different sites [3], which creates a larger dependency and communication network than in collocated projects;
2. Delay in communication and coordination: communication and information exchange take longer time in GSD than in collocated projects [3, 23];
3. Limited choice of coordination mechanism: distances among sites disable team coordination in face-to-face manner [9];
4. Difficulty on identifying coordination requirement: identification of people who are technically inter-dependent is difficult [24, 25];
5. Difficulty in scheduling task: the involvement of many sites across geographical locations and time zones makes scheduling common meetings difficult [26];
6. Lack of trust: it is harder to establish mutual trust among project stakeholders in GSD than in collocated projects [20, 27];
7. Lack of team awareness: GSD team members are often unaware of tasks and activities of other developers from remote sites [23, 28];
8. Misinterpretation: team members can misunderstand each other working in different environment and contextual information is not given sufficiently [9];

While some of these challenges can be accounted for a specific type of global distances, i.e. geographical distance lead to lack of face-to-face contact, most of the challenges with team coordination are often caused by many reasons and hard to identify the root cause [9]. In this study, we focused on coordination challenges created by organizational boundary attributes. To best of our knowledge, there is no empirical study that comparatively and systematically investigates the influence of different attributes of the organizational boundary on team coordination before.

2.3. Effective coordination mechanisms
Dependences among tasks and task holders are handled by using coordination mechanisms, which are practices, methods, strategies or policies that are used to manage the dependencies among team members and tasks [21]. Previous studies suggests several effective coordination practices for inter-organizational software projects:

1. Synchronization of main milestones [29]: customer and vendor teams keep their own development processes and only main phases and milestones were synchronized;
2. Frequent deliveries [29]: deliveries are integrated and tested in a daily basic;

<table>
<thead>
<tr>
<th>Table 1: Data collection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No. of interviews</strong></td>
</tr>
<tr>
<td><strong>Interview roles</strong></td>
</tr>
<tr>
<td><strong>Interview across site</strong></td>
</tr>
<tr>
<td><strong>No. of observations</strong></td>
</tr>
</tbody>
</table>
3. Establishment of peer-to-peer links [29]: contact points for manager-to-manager and developer-to-developer;

4. Early relationship building [29]: having involved personnel meet face-to-face in beginning phases of a project

5. Early architectural assignment [25]: assignment of task corresponding to architectural modularization;

6. Shared management and development tool [30]: having the same set of tools across sites;

7. Boundary spanner [31]: using an unofficial role of boundary spanners as a explicit coordination mechanism

This list is used as a basic for exploring and comparing effective coordination mechanism in our cases.

3. Research design

3.1. Study design
We investigated two GSD projects involving four different organizations. The reason for choosing these projects was that (1) the projects involved GSD and has faced with coordination challenges, (2) the projects represented different organizational relationship, and (3) the projects offered availability of rich data sources and multiple points of view. The unit of analysis is a development team, which matches with the objective of exploring inter-team coordination activities.

3.2. Data collection and analysis
We relied on data triangulation as defined by Yin [32]. Our data collection was based on:

- Interviews as a primary data source
- Documentation, and passive observation as a secondary data source

We conducted 10 semi-interviews (3 of them conducted via telephones) with different project roles, such as project manager, technical leader and developer. Interview guide consists of 4 main parts: (1) project and interviewee background, (2) general challenges when collaborating with others from distance, (3) specific problems related to organizational differences when working with other teams, (4) best practices when working with teams from other organizations. The interview guide was designed using the checklist from the literature review. Each interview lasted from 50 to 75 minutes. The interviews were recorded and then transcribed.

Two observation sections were performed in Case A, as given in Table 1. The first observation focused on a 4-hour work of a team leader, the later one focused on a 4-hour workshop between Team A1 and A2 (section III.C).

Figure 2: Data analysis process
Table 2: Cases context

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Case A</th>
<th>Case B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project type</td>
<td>Bespoken</td>
<td>Market driven</td>
</tr>
<tr>
<td>Global arrangement</td>
<td>Outsourcing</td>
<td>Insourcing</td>
</tr>
<tr>
<td>Project size</td>
<td>Medium</td>
<td>Large</td>
</tr>
<tr>
<td>Project age</td>
<td>5 years</td>
<td>5 years</td>
</tr>
<tr>
<td>Product</td>
<td>Ship management system</td>
<td>Search engine system</td>
</tr>
<tr>
<td>Involved organization</td>
<td>A customer team and two software vendors</td>
<td>A headquarters and two offshore sites</td>
</tr>
<tr>
<td>Team size</td>
<td>13</td>
<td>150</td>
</tr>
<tr>
<td>Inter team dependencies</td>
<td>Technical</td>
<td>Technical, temporal and process</td>
</tr>
<tr>
<td>Site locations</td>
<td>3 sites within Norway</td>
<td>3 sites in USA and Norway</td>
</tr>
</tbody>
</table>

We collected 49 A4 pages of interview transcripts and 8 A4 pages of observation notes as a major data material. Other materials such as email exchanged, company website, product and project specification was collected for understanding case context. All transcripts were sent to interviewees for confirmation.

We analyzed the data in several steps. Firstly, we read all interview transcripts and relevant documents, and coded them according to open coding [33]. Each segment of text that expresses either organizational boundary, team coordination problem or effective coordination mechanisms were labeled with an appropriate code. Secondly, we compared segments from different interviews, but from the same case, that were assigned the same code (axial coding).

In axial coding, indicators and characteristics for each concept were searched for, in order to define that concept [33]. Finally, the concepts were compared between two cases and the differences were discussed. The overall process is described in Figure 2.

3.3. Case contexts
This section describes a context of each project, including products, team structure, coordination practices and the need for inter-team coordination, summarized in Table 2.

Context in Case A:
The project studied in Case A is a ship management system. The project involves two external contractors, so-called Team A1 and Team A2. Team A1 locates in Trondheim, Norway and implements the web application systems. The internal coordination practices include face-to-face discussions, informal meeting, team foundation server (TFS) and adoption of a lightweight task tracking system. Task assignments are conducted based on early architectural division.

External communication with customers is mainly done by phone and email. A few times a year, the Team A1 team leader visits external stakeholders (customer team and Team A2). Team A2 locates in Stavanger, Norway and implements the desktop application. The development methodology is Scrum. External collaboration is done via email, and frequency of email exchange varies according to the needs of the project. The communication between Team A2 and external stakeholders is based on phone, face-to-face meeting, email and TFS.
Team A1 and A2 are responsible for the two different parts of the systems, but sharing a common data access layer and database. When someone develops or modifies a feature that needs to update the database, the change had to be informed to the other team for updating affected features. There was little dependency between two teams in the beginning of the projects. At the time this study was conducted, both teams were performing a joint effort on refactoring overlap code base and updating common data model. For developing the new feature, Team A1 had a resource dependency on Team A2 as a key member of Team A2 knew much about business logic of the whole system. Both teams had a technical dependency on each other as they developed their part in parallel and need to synchronize these parts and keep the data model consistent.

Context in Case B:
The project in Case B is a search engine system as part of a large IS management system. The search engine system has many API interfaces, which are used by many higher-level systems. The part of the project that we investigated only relate to a team in the headquarter office in USA and two development centers in Norway, namely Team B1, B2 and B3, correspondingly.

Team B1 locates in the headquarter office of Cooperation X in USA. X is a large software production organization with established software development processes and many years of experiences in GSD. Common set of collaboration practices are used among Team B1, B2 and B3, which are daily (virtual) meeting, teleconferencing meeting, email and TFS. A few times a year, there is an exchange of team members between USA and Norway. Team B2 and B3 were initially two branches of a Norwegian company, developing a standalone search product. After acquired by an US company, Team B2 and B3 became a part of development centers in Norway. In addition to common collaboration practices for the whole cooperation, Team B2 and B3 remains local collaboration practices, such as communication via Git, informal chat and frequent visit between Team B2 and B3.

Collaboration between B1, B2 and B3 is a basic need for developing the main system. There is high amount of technical, temporal and process dependency among these teams. Each team develops some modules that will be integrated later in the process. The synchronization of tasks is crucial to achieve overall project milestones and quality of the whole search system.

4. Findings
This section presents attributes of organizational boundary that contribute to inter-team coordination problems. Table 3 presents the organizational boundary attributes and its consequence on coordination and development activities. Besides, we also describe how the coordination mechanisms were used to close this gap. Each aspect is illustrated with quotes from the interviews.

4.1. Findings on Case A:

A.1 Collaboration policy
In this case, collaboration policy refers to an established governance policy by customer and competition attitude of each team towards other organizations. Firstly, customer governance policy did not support inter-team coordination on tightly inter-dependent tasks, which
contributed to problems of lack of inter-team awareness, technical debt and source code replication between two teams.

Respondents from both organizations and in both roles, developer and manager, were aware of a tight control mechanism over collaboration between two teams in the beginning of the project. A developer from Team A2 stated that until last year, inter-team communication is mainly done via customer. Any necessary technical discussion was controlled by the customer team. Another developer from Team A1 also said: “…the product owners didn’t wish us to talk to [Team A2] without the presence of the … This was based on the kind of suspicions that we, as two consultant companies, would make stuff up to make some money.”

Both teams had worked almost independently until the coordination needs had extrapolated to what the customer team was not able to manage. As each team worked on their own codebase, they unexpectedly created redundant code on data access layers and some utility functions. The overlap source code became larger and larger overtime, resulting in a technical debt, which needs to be resolved before starting a new highly joint development tasks.

Later on, the governance policy on collaboration between vendor teams had been loosened, partly because of the long-term relationship between customer and each team had been established. Both teams could have meetings or discussions without customer representative. However, we still observed a low level of awareness across organizational boundary. Both team did not know much about how the other team’s status and work progress. Two interviewed developers from Team A1 were not aware of status of Team A2 and two other interviewed developers from Team A2 were also not aware of status of Team A1: “Researcher: Do you talk to Team A2 about progress and status of the work? DevA1: …not that much as we should. We are very autonomous in this aspect. But we do talk a lot with [Customer names] about progress, not much to [Team A2]…”

Secondly, the competition attitude to each other also created a barrier for team coordination. Each team belong to a software vendor with similar capability in the market, who are directly competitors in getting contracts from the customer. The better relationship and perceived credits from the customer could lead to more contracts for each team. The attitude of competition was recognized from both Team A1 and Team A2. A technical leader from Team A1 said that: “Regarding to the organizational dimension, it is not a problem with legal issues. But they do collaborate as competitors, so we need to keep the competition capability”. One developer from Team A2 also stated that: “…so we are in the competition with [Team A1]. Maybe [Team A1] want to do it too so we need to discuss and make argument about why should we do it instead. This is absolutely one part of the competition. But we also want to look good compared to [Team A1]. And we know they want to do the same …”

Although this attitude was positively perceived by the customer since it motivated both teams to perform better and achieve higher level of customer satisfaction, the competition attitude limited contextual information sharing, knowledge sharing and the teamness between two teams. As recognized from developer’s perspective, there was a “comparison mindset” towards the other team, in which knowledge domain and customer recognition were competitive advantages: “We enjoy working in a way that satisfy customer because we know we gain something compared to [Team A2] and the other way around for [Team A2]”. By reading emails exchanged between Team A1 and Team A2, we observed that little
information related to customer’s status and project information were exchanged. Most of the time, both teams discussed about technical contents.

Besides, competition attitude hindered knowledge transfer between two teams. Team A2 adopted a strategy to discuss about business logic of a joint developed features just-enough and just-in-time. One developer from Team A2 said: “[Team A1] relies on us 100% because they do not have business domain knowledge. If some of us is on vacation, it is kind of not having control in this case and high effect on others...”

Moreover, the feeling of working as a big team was not observed from both teams. Across different interview transcripts, the used pronouns were “we” versus “they” when talking about team collaboration. The customers’ name was mentioned from interviewee from both sides, but there is no private name of person from other team found in the transcripts: “…Usually you don’t ask other developers from another company, which you don’t know him well. You don’t ask him first even though you probably know that he is the one who knows the answer. Firstly you tried yourself and then you ask someone locally.”

**A.2 Team organization**

Team organization refers to challenges related to differences in team structure creating on team collaboration. We explored which coordination problems arise when a team in a flat structure (Team A1) working with a team in a hierarchical structure (Team A2). As architecture of the system is clearly modular from the beginning, both teams have quite clear division of technical task and responsibility. However, the mismatch in code submission structures introduced difficulty on team communication. In Team A1, team leader did most of the contact and receiving tasks. A task was broken down and given to the developers but they are not allowed to check in directly to a common VCS system. The team leader spent significant amount of time on delegating and integrating tasks in the team. He also performed some tests before checking in. In Team A2, developers had more autonomy on checking the code into the common VCS system. Therefore, for team A1, it was easy to find the relevant person from Team A2 who submitted a piece of code or a bug report. However, developers from Team A2 had to wait for information forwarding back and forth from the team leader in Team A1. As mentioned by a developer from Team A2: “…if it does not compile you can send an email (to the team leader of Team A1). I would see what the check in comment is. I will do it before I contact him …”.

**A.3. Engineering process**

Engineering process refers to the challenges of team coordination created by the variation on development processes. Lack of frequent and in-depth coordination during task integration led to complexity and extra efforts on task integration.

Team A1 and Team A2 had different approaches on capturing requirements, planning tasks, implementation and integration of the tasks. Team A2 adopted SCRUM with many small sprints while Team A1 used ad hoc and informal agile development, without splitting tasks into sprints and continuously receiving and developing tasks. The different development approaches hindered in-depth discussion on technical tasks at team level, as one developer said: “…we work in such different ways so we don’t share work processes. We communicated in a bit higher level than what we are doing now. As we see thing from Scrum, we have more defined tasks for near future and backlog items, but [Team A1] they could give feedback on how they are doing continuously”. It also limited establishment of frequent discussion among
developers across sites. Coordination was driven on deadline basic rather than daily basic: “…I am not very sure about [Team A1]’s deadlines. But they have kind of implicit deadline by that most of their system need to be ready for us …”.

The difference on development iteration duration made teams harder to find common milestones for task integration, as mentioned by the team leader of Team A1: “…If you work for one branch for half of a year, there is quite a lot of code to merge. [Team A2] is ready to merge every 2 months. … [Team A2] is ready to launch every month but it costs too much to ensure the quality…” Both teams had different set of internal deadlines regarding to the project. Inter-team coordination needs increased when Team A1 need to deliver a function that related to Team A2 and vice versa. However, at that time, Team A2 might not give much resource for cross-team synchronization activities, which lead to some issues as communication delay and difficulty on meeting scheduling.

A.4 Development practices
Variation on testing and coding approaches between two teams introduced mistrust and extra coordination effort. The difference on code and comment style was mentioned by one developer in Team A2. The developer experienced some difficulty on understanding code from Team A1 due to insufficient comments, lengthy and complicated code files. He suggested that: “If all developers don’t comment or clean over it you experience kind of low quality. We have established better mutual understanding about what good code is because we shared a lot of high-level source code. I am sure if we do a lot of high level code we would be able to come to common understanding of how it should be done”.

The difference on testing approaches also seemed to create a mismatch on expectation of the quality of the integrated codes. Automated testing reached 100% coverage in Team A1. Team A2 had only done automated testing partially. This introduced some topics for discussion between the teams when a bug with cross-boundary impact appears. The awareness of variation in testing approaches between two sites seemed to increase confidence of Team A1 when discussing on quality of code from each site.

Tool adoption refers to how configuration management and development tools were used in each team. Shared set of development infrastructures without a common set of adoption practices led to unnecessary coordination effort. Although both of teams shared the same configuration management tool, the way they used these tools were quite different. In Team A2, the functionalities of TFS were fully adopted, from recording requirements and modification requests from customers, storing source code, implementation and tracking and bug fixing. In Team A1, TFS was not actually adopted for communicating with customer and Team A2, as mentioned by a developer from Team A1 reported that: “TFS have a lot of features for these kinds of things. We tried it out and we are kind of in another self-organizing way. We found it is too much bureaucracy for our way of working…”

The only function of TFS shared by both teams was source code control. Therefore, some features of TFS that is beneficial for inter-team coordination, such as “gated check-in”, which helps to enforce a clear contract between developers and avoid manual check of problems with logical source code structure, data type, variable naming, variable scope and error handling, were not used. A technical leader of Team A1 highlighted that the common process and practices are important for working over distances: “If you establish guideline for coding,
you have a lot of tools you can achieve the quality you want in the project. And it should not depend on whether you seat on other country or city or whatever.”

A.5 Effective coordination practices
We also observed an adoption of effective coordination mechanisms, such as establishment of personal relationship among developers, early and clear architectural assignment and boundary spanner.

Informal communication among developers was gradually introduced as personal relationship between developers was built. One developer from Team A1 said, “We actually share everything and work even more together. Whenever we (internal people) talk together, before we met [Team A2], we always think they are so creepy… But now it has changed quite a lot by meeting them and seeing them just like us. It is a big gain for us.” Besides, architectural assignment helped both teams limited the area need to be coordinated. The contract-based division of work was based on the overall architecture of the whole system. Therefore, it reduced the number of common parts between two teams.

The project leader in Team A1 was considered as an unofficial boundary spanner and also an effective coordination mechanism. For an internal team member, boundary spanners were seen as a closest position to external stakeholders and have influence on them. A developer in Team A1 said, “I believe that the project owner trust us deeply. They trust so much that letting [Project leader Team 1] as an assumption part of technical insight. If [Team 1] exploded and disappear the customer will have problem. I don’t think [Team 1] is easy to replace at this time”. For a customer, boundary spanners were seen as a source of information and also a target for giving feedback and negotiating tasks.

4.2. Findings on Case B

B.1 Collaboration policy
In case B, collaboration policy refers to different expectation on task responsibility on borderline tasks. The team’s divergent expectation led to extra effort on team coordination and conflict solving. Team B2 and B3 pursued a relatively high level of autonomy in collaborating with other development centers. The divergence between Team B2 and other teams was caused by task dependency and unclear assignment of responsibility at team level. As mentioned by a program manager in Team B2, each team often had a special competition on trying to not to take more tasks: “We had so much to do that we tried to push the tasks in the borderline between team. We tried to push it to other team”.

The given reason was that Team B2 wanted to focus on their modules and not to spread through many parts of the system. The coordination occurred in these borderline components and conflicts also appeared here. The program manager from team B2 mentioned that: “When there is a problem with the exchanged component, the other team can work around or we can do to fix the issues... At the moment we have more problems with who is going to do what.” This question was rather a strategic question that is resolved at management level: “… normally, it (the conflicted tasks) is a negotiation between teams by ourselves. But if we don’t have the agreement, we need someone to make the prioritization.”
As mentioned by the program manager in Team B2, most of his work time spent on coordinating inter-team activities, in which, solving the conflict related to tasks, work process and temporal dependency are the most time consuming. It seemed that a clear responsibility and ownership of a component could facilitate team coordination and helped to avoid such conflicts.

**B.2 Team organization**

There was no significant problem related to communication across team in Case B. The communication occurred much more frequent on management level rather than it did in developer level. At the time of this study, Team B1 and B2 were putting a lot effort on enhancing the code commit structure. In general, the branching system was organized by the organizational structure. However, allowing the development of branches for each developer led to a complex branch system. When the integration was not frequently performed, the different branching models across sites might be problematic. A developer from Team B2 experienced extra coordination effort when merging branches from Team B1. The habits on working with Git facilitated the development of shared understanding among developers between Team B1 and B2. However the technical integration with a team outside Norway was more difficult. The program manager from Team B2 mentioned the problem: “So when the developers check in (to the main branch), should they be free to check in any time or what kind of testing we need to do before checking in? We are adjusting this process.”

**B.3 Engineering process**

Recalling challenges with incompatible development processes, difference on development process caused an unexpected long duration of knowledge transfer. When acquired by X and started collaborating with Team B1, Team B2 and B3 had a difficult time on working with the US site. Although the acquisition process had gone through a process compatibility assessment to support the integration of the new development center to the cooperation, there were a lot of challenges in the beginning.

Firstly, developers from Team B2 and B3 had to adapt to the new introduced engineering process, tools and practices. A developer from Team B2 mentioned: “… when we were acquired by [Cooperation B] is a big difficulty for us to understand what they are talking about, what is the terminology, what is [Work package name] is. [Work package name] is not the way we are working before. We did have things in Git and exchanged like connection to Git, not sending the [work package name] …” As mentioned by the program manager from Team B2, some parts of the old development process still remains due to the given autonomy of the development center.

Secondly, Team B2 and B3 experienced that the different focus on quality and productivity led to difficulty on coordinating technical tasks. Team B1, and the whole cooperation had a high focus on quality of software product, since their work serves for global market with a large amount of customers. Team B2 and B3 were used to work with a small set of local customer, hence focusing more on finishing on time and customization. The different focus led to different development and testing approaches between the two teams. The program manager from Team B2 said: “The new focus on quality was very difficult… We had to spend more resources to have better quality in this case. Earlier we had a shorter last LOC developed to the release date. Now we have a longer time and an established test process. For [B2, B3’s old company] just when the last line compiled we send the package to the customer and see their feedback.”
B.4 Development practices

Issues with different coding style, naming convention and error handling were automatically checked by tools before integrating the check-in source code into a main branch. Therefore, most of variation tool adoption and practices were mitigated in this case.

Variations on test automation among team B1, B2 and B3 introduced the mistrust between teams. The automation testing coverage rates also varied among teams. A typical development team was encouraged to increase the coverage rate up to 80%. A developer from Team B3 complained about the quality of code from other teams: “… we also work with a team in Sri Lanka. There is a huge difference in quality between the two, but quality levels will converge next year. They will implement more automation test …”. While Team B1 focused on developing automated tests, Team B2 did not consider high coverage of test automation as an indication of high code quality. As Team B1 and B2 worked relatively close to each other, this issue had been extensively discussed among program managers.

B.5 Effective coordination practices

We observed the effective adoption of many coordination mechanisms, given in Table 4. All teams followed the same global development process, which consists frequent deliveries, synchronization of milestones and a consistent set of collaboration tools. On the collaboration approaches, all program managers didn’t use desk phone but use a unified communication platform. A developer in Team B2 mentioned that: “… for daily meeting it is like people seat in their own office place with their headset. I have my colleague who seats behind me. Very often we participate in the same meeting back-to-back and talk to other locations. I would say it is life safety for us to do collaboration with other locations.” These approaches helped to form a similar organizational culture and common process between the teams.

Relationship building was established by having staff exchange among locations. The program manager from Team B2 mentioned that they have some short-term visit every year. A developer from Team B2 said, “Researcher: do you have some informal talks and social activities with other teams? DevB2: when I was in Oslo, I spent two evenings with dinners. We had a Christmas party. A lot of us are friends outside work”

A board of program managers acted as official boundary spanners that handle cross-team collaboration issues. The program manager from B2 mentioned that 50 – 80% of his work time was spent on coordinating inter-team activities, including communication of task status and schedule, conflict resolution and management of dependencies among locations. The specialization of the board as a coordination mechanism reduces the workload on communication in developer level.

5. Discussion

We have described, in Section IV, the results from data collected on two cases that faced challenges in coordinating technical tasks. We now discuss the cases in light of our RQs.

RQ1: What are attributes of the organizational boundary that introduce challenges on team coordination in GSD?

On the collaboration policy, we observed a direct impact of high-level governance policy on coordination of technical tasks. Developers often collaborate in a multiple dimensional
context, where not only technical dependency matters, but also resource and strategic dependencies does. In case A, customer’s collaboration policy enlarges the organizational distance between two firms. In case B, high-level management policy clashed on autonomy of each development center. The misalignment of expectation and commitment among teams about responsibility of their partners were not clear due to the large organizational distances. In both case, the collaboration policy inhibited synchronization activities. This is a well-known problem and a common advice is to address who-do-what question in collaborating developers from other locations [15, 23, 28]. In our cases, the alignment of collaboration policy with coordination needs is seems to be equally important and need to be addressed at first.

On the team organization, alignment of communication structure and code submission structure is a challenge in both cases. This observation is surprising as team organization issues still appear in a small project with relatively little coordination needs. It seems that working on branches and committing code in a shared repository require a large amount of coordination. Previous literature often highlights the importance of supporting communication between developers who work on the same tasks [4, 11, 15]. Our case reveals that in a fine-grained level, the communication structure among developers should be aligned with their code commit structure.

On the engineering process, we found that differences on development methodology, such as sprint iterations and team’s prioritized focus hinder frequent team communication and task synchronization. Even in the same organization, the coordination issues created by procedure differences should be expected. Paasivaara et al. reported that synchronization of milestones could help to coordinate two different development processes [29]. However, our observation suggests that there should be at least some common parts of processes. Depending on the coordination needs, this procedural commonality should allow synchronization activities across teams without extra resources and efforts.

On the development practices, we confirmed that the variation on coding and testing practices introduce the negative perception on the work of others and hinders the establishment of common understanding [19, 20]. Differences on testing and coding practices can introduce difficulty on establishing common technical understanding. In our case, we observed coordination problems in a homogeneous development and collaboration infrastructure. In case A, some features in TFS could reduce the needs of communication if they were adopted in both sites. In case B, synchronization of source code in both Git and TFS introduced some amount of unnecessary extra works. The coordination problem is more severe in case A as the organizational boundary is larger than that in Case B. This aligns with findings by De Souza and his colleagues, which highlights the importance of adoption of collaboration tool across geographical boundaries [15].

RQ2: What are the coordination mechanisms that help to mitigate organizational boundary?
In this section, we discuss the effectiveness of some common mentioned effective coordination practices in previous studies [25, 29, 30, 31]. In Table 4, the coordination mechanism is either effectively used (marked by +), or not effectively adopted (marked by -) or not used (marked by O). Synchronization of main milestones were observed in both cases. In case A, the duration between two continuous common milestones was too long to sufficiently exchange information and intermediates. Attempts to coordinate in a higher
frequency were hindered by different deadlines and priorities of each team. Therefore, the effectiveness of this mechanism depends much on planning common milestones among teams without affecting each team’s priorities.

Frequent deliveries were successfully adopted in Case B, as developers in all teams worked on daily build. This practice is very important for avoiding divergence on codebase and accumulation of technical debts. However, it was not applicable in Case A since it would require a global development processes for both partners.

Establishment of peer-to-peer links: we didn’t notice the establishment of developer-to-developer links. The combination of manager-to-manager and manager-to-developer contact is a common pattern in both cases. Inter-team awareness and the feeling of teamness can be improved by facilitating developer-to-developer communication.

Early relationship building: our data showed that knowing other team member in person is especially important for coordination to succeed. Early face-to-face meetings and some informal talks should be done to facilitate later electronic communication. However, establishment of social relationship is initially limited by the control policy in case A.

Early architectural assignment: in both cases, task breakdown structure was created to support team organization structures. The source code branch and repository structure is balanced between team structures and product structures. For complex projects like Case B, this is compulsory to reduce complexity and inter-dependency between teams and tasks.

Shared management and development tools: were established in both cases. However, the full advantages of this are disabled in case A, due to the variety of development practices. Hence, we suggested that common tools set should be accompanied with common usage practices in order to fully utilize the advanced development environments.

Boundary spanners played an important role in supporting team coordination in both cases. The team leader (in case A) and the program manager (in case B) were the ones who know about business domain, technical insight of the product and established a stable relationship with the customer teams. Therefore, they no only facilitate team communication, but also to transfer knowledge and to contact customers.

6. Conclusion

Distances do not need to be global to matter, and not all problems in GSD are caused by physical distances. From two exploratory case studies, we observed several team challenges with coordinating across organizational boundaries and a set of effective coordination practices.

<table>
<thead>
<tr>
<th>Table 4: Team coordination mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team coordination mechanism</td>
</tr>
<tr>
<td>Establishment of peer to peer links</td>
</tr>
<tr>
<td>Frequent deliveries</td>
</tr>
<tr>
<td>Synchronization of milestones</td>
</tr>
<tr>
<td>Shared management and development tool</td>
</tr>
<tr>
<td>Early relationship building</td>
</tr>
<tr>
<td>Early architectural assignment</td>
</tr>
<tr>
<td>Boundary spanner</td>
</tr>
</tbody>
</table>
To response to “What are attributes of the organizational boundary that introduce challenges on team coordination in GSD?”, we found that collaboration policy, organization structure, engineering process, development practice and tool adoption negatively influence technical coordination among project stakeholders. There is variation on specific attributes of these organizational boundaries and the extent of their influence on team coordination, depending on the coordination needs, project size, and type of organizational relationship. For practitioners, our study showed that firstly, project managers need to be aware of the level of development process variation among organizations for the project coordination between organizations to be successful. Secondly, when determining high-level of collaboration policy, the indirect influence on team coordination at technical level should be considered. Lastly, after obtaining shared development environments, a guideline for shared standard and tool adoption practices should be established.

To response to “What are the coordination mechanisms that help to mitigate organizational boundary?”, we confirmed the effectiveness of practices, such as early relationship building, early architectural assignment, and knowledge broker in both cases. The large organizational boundary inhibits the effectiveness of synchronization activities, frequent deliveries, peer-to-peer contact, and shared management and development tools. For practitioners, our study confirmed that relying on a single coordination mechanism is not sufficient to support team collaboration beyond organizational boundary. The formal processes, standards and guideline should be established among organizations to facilitate the transparent collaboration process in both management and operation levels. Most importantly, the role of boundary spanners in supporting team coordination need to be recognized and supported.

In future works, we will continue to explore the other aspects of organizational boundary in GSD. Further studies in this direction can focus on identification of some measures of organizational distance and quantitatively investigate their relationship to team coordination.

References


Paper MP6 - On the role of boundary spanners as team coordination mechanism in organizationally distributed projects

On the role of boundary spanners as team coordination mechanisms in organizationally distributed projects

Anh Nguyen Duc, Daniela S. Cruzes, Reidar Conradi
Norwegian University of Science and Technology
Trondheim, Norway
anhn, drcuzes, Reidar.Conradi@idi.ntnu.no

Abstract—Effective team coordination is crucial for successful global software projects. In projects that present significant global boundaries among sites, boundary spanning is an important team coordination mechanism. This paper provides an understanding of how boundary spanner role, boundary objects and boundary spanning activities are adopted to resolve coordination gaps among organizationally distributed development team. The qualitative data were collected and compared among four types of global software projects. The use of boundary spanning activities is compared among four different cases to identify the challenge and practices of team coordination. We found that boundary spanners take an important role in mediating task dependency and information dependency across boundaries. Besides, we identified four ways that boundary objects can be used to constitute, complement or back up for boundary spanners.

Keywords: global software development, case study, boundary spanning, team coordination, boundary object

1. Introduction

Recognized as an increasingly popular phenomenon in software production industry, global software development (GSD) has received significant interest from research and practitioner in the last decade. Although many collaborative development infrastructure [1, 2], processes [3] and practices [4, 5] has been proposed to facilitate global software projects, GSD continues to face many challenges created by multiple dimensions of global boundary. An industrial report showed that approximately 50% of offshore projects fail to deliver the expected benefits [6]. It is also showed that 60% of empirical studies on GSD are problem reports and failure stories [7], and 58% of these empirical studies showed a negative influence of global boundaries on team performance [8].

Team coordination, understood as activities to manage dependencies between tasks and task holders, is recognized as a main challenge for success of global software projects [9]. Compared to collocated software projects, a distributed team needs to handle not only the uncertainty of requirements and technical complexity of development tasks, but also the interdependencies between tasks and task holders across geographical, temporal, cultural and organizational boundaries. Geographically dispersed team members often lack mutual knowledge and common context, leading to misunderstanding and breakdowns in communication and collaboration [10]. They also have problems with different working processes and practices [11], lack of trust [12], and difficulty in sharing knowledge and expertise [13].
One of the mechanisms that organizations employ for addressing these boundary issues is to rely on individuals who obtain the role of linking the organization’s internal networks with external sources of knowledge and information. This role is referred differently as “liaison”, “knowledge mediators”, “information broker”, “bridge engineer” in Information System (IS) and Software Engineering (SE) literature [14-18]. In a software project, the boundary spanner role can be taken by a requirement analyst, a project manager or special engineer to cross boundaries between customers and development teams or between different vendors. Formally nominated or informally emerged, boundary spanners are intended to facilitate knowledge sharing, transporting information, translating requirements and transferring expertise [19]. Boundary spanning activities are found as a significant influenced factor on team collaboration across boundaries and a predictor of several different organizational outcomes such as knowledge sharing, team coordination and team effectiveness [20].

In spite of the presence of boundary spanners roles, literature on SE revealed a number of failure stories on team coordination breakdown [22, 23]. Therefore, it seems that the role of boundary spanners and their effectiveness as a team coordination mechanism varies widely and depends on many individual and organizational factors. In the context of GSD, the influence of boundary spanning activities and objects on team coordination is little explored. This study continues our works on challenges and solution for team coordination in GSD. Our prior studies had investigated organizational boundaries in software ecosystem, coexistence of competition and collaboration attitude [24] and influence of team coordination on project outcomes [11]. As the next step, this study addresses the questions “How boundary spanning support team coordination in organizationally distributed teams?”.

The remainder of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the research method, followed by the results in Section 4. Section 5 presents a discussion and Section 6 concludes the papers.

2. Related Work

2.1. Organizational boundaries and team coordination

The term “team coordination” is defined as “activities required to maintain consistency within a work product or to manage dependencies within the workflow” [9]. Several types of dependencies were identified, such as: task dependencies (i.e., needed to integrate software parts seamlessly), temporal dependencies (i.e., necessary to synchronize activities and adhere to project schedules), software process dependencies (i.e., required to adhere to the established software processes and act as agreed with other team members), and knowledge dependencies (i.e., a team needed business domain knowledge from the other team to accomplish a task) [9, 26]. Failure to support these dependencies results in a coordination breakdown, such as: lack of team awareness, lack of task awareness, lack of mutual trust, limited choice of coordination mechanism, difficulty in task scheduling, delay in communication, lack of common understanding [8]. In this study, we used these challenges to evaluate the effectiveness of team coordination. Perceived coordination effectiveness is achieved when all team members do not feel the presence of any mentioned problems.

2.2. Boundary spanner roles, boundary spanning activities and boundary objects

Empirical work on boundary spanning has focused on three important aspects of boundary spanning, namely boundary spanner, boundary objects and boundary spanning activities [27, 28]. Early work viewed the boundary spanner as a role linking “organizational structure to
environmental elements, whether by buffering, moderating, or influencing the environment” [29]. Boundary spanner roles can be occupied by different team members, such as project managers, technical leaders, requirement analysts and external coordinators [14-18]. Di Marco et al specified that boundary spanners are not necessarily formal team leaders or project managers, but can be any of the team members who connect the members of distributed sub teams in the project network [17]. Levina et al. studied offshore collaboration in information systems development and found that a middle manager on the onshore team can act as a boundary spanner to mediate the negative effects arising from status and cultural differences [30].

Several boundary spanning activities category has been proposed in IS literature. Levina et al. revealed that boundary spanners-in-practice must actually engage in boundary spanning by relating practices of one team to practices of another team and negotiating the meaning and terms of the relationship [19]. Carmel and Agarwal found the ability of boundary spanner to cross cultural boundary by resolving cultural gaps and translating linguistic information [31, 32]. Pawlowski & Robey 2004 described the ability to facilitate sharing of expertise, integrating communication of the team [33]. Faraj and Yan 2009 highlighted the ability to protect inner team and obtain political support [34]. In our context, we defined boundary spanning activities as mechanism that (1) bring team members into contact with non-members or (2) overcomes knowledge and information boundary by engaging agents from different knowledge communities in collective activities.

In addition to spanner, boundary objects can be used to coordinated team member’s activities. The concept of boundary objects, developed by Star [27] described artifacts that “sit in the middle” of diverse knowledge groups, establishing a “shared and sharable” context for distributed problem solving. These artifacts need to be “both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identity across sites”. The term “boundary” in “boundary object” can refer to a difference between a group of developers vs. a group of tester, a group of newcomer in OSS projects vs. a group of a core developer, a group of customers vs. a group of requirement analyst. Boundary objects are not the result of team collaboration, but a means to facilitate it. Nicolini et al. provide a pluralist view to highlight three types of objects (1) an infrastructure that enables team collaboration, (2) a transportation devices for cross-boundary work and (3) a trigger for team collaboration [35]. In the context of IT systems, boundary objects can be document archives, source code, design artifacts, physical prototypes, design documents and repository [30, 33]. Most GSD-related artifacts are versioned and reside in local file repositories. In this study, we focused on the use of artifacts commonly adopted from both sides of the team to support "cooperating transactions" [44].

3. Research Methodology
3.1. Research questions and an initial conceptual framework

Literature on boundary spanning often distinguishes between formal and informal boundary spanners [19, 30]. We expected both types are having the same set of capacities in supporting team coordination. Therefore, as a first step to understand boundary spanning as coordination mechanism, we would like to identify the common capacities of a boundary spanner. Although several frameworks exist to classify boundary spanning activities [31-34], little has been known about how the boundary spanning activities are performed to address
interdependency among team members across global boundaries. Besides, boundary object is another team coordination mechanism that has received little attention. We would like to identify the role of boundary object in supporting team coordination. This leads to our three research questions:

- RQ1. What are common capacities of a boundary spanner as a project coordinator in organizationally distributed software projects?
- RQ2. In which way a boundary spanner influence team coordination in organizationally distributed software projects?
- RQ3. What is the role of boundary objects on supporting team coordination in organizationally distributed software projects?

A general coordination framework shown in Figure 1 is useful for positioning the present research. This framework is an extension of one previously proposed by Espinosa et al. [36]. The author links different global boundary variable, team coordination mechanism and project outcome concept in an input–process–output model. The framework includes three coordination mechanisms, namely task programming, team communication and team cognition. In our study, we focused on boundary spanning as a focus on team coordination mechanism.

3.2. Case study design

Case selection:

With the purpose of understanding successful boundary spanning activities and objects, we performed an exploratory study [37]. The unit of analysis for each individual case was a global software development project, each project involved more than one organization. Due to agreements about confidential information with interviewee participants, we used aliases instead of real project identity, called Project River, Project Mountain, Project Ocean and Project Tree.

Case selection has two objectives: (1) to obtain a representative sample and (2) to demonstrate variation on the dimensions of theoretical interest. Firstly, to represent different global boundaries, we selected Project River, Project Mountain and Project Ocean that represents for different global arrangement setting, such as insourcing, offshore outsourcing and inshore outsourcing [38]. To compare the interaction between boundary objects and boundary spanner, we selected Project Tree as an extreme case since it is inherently different from the other three projects (mainly relied on set of boundary objects). Secondly, to be able to reason about influences of different global setting on team coordination, we need to select cases with similar set of coordination mechanisms and practices. Therefore, all four cases adopted some common coordination mechanisms, such as configuration management system, email exchange, site visiting, and social relationship building.

![Figure 1: Initial conceptual framework (adapted from [36])](image-url)
Certain practical consideration also influenced case selection. As our researchers locate in Trondheim, Norway, Project River, Mountain and Tree has a site located in Trondheim, Norway. In Case Ocean, data was collected conveniently from our researcher’s professional networking. The selected projects needed to last long enough to ensure its maturity and also enough data for conducting the research. The selected projects needed to be ongoing so the stakeholders can recall and describe current events. Besides, we maximized variety across cases to provide better concept definition by considering a variety of contexts and factors in the emergent theoretical concepts [39]. Projects were selected with different product types, product domain, team size and team organization. Key details of these four cases are summarized in Table 1.

**Data collection:**

We relied on data triangulation as defined by Yin [40]. Our data collection based on: (1) interviews as a primary data source, (2) Documentation, and passive observations as a secondary data sources. We conducted 16 interviews with different project roles, such as project manager, technical leader and developers. The interviews were recorded and then transcribed. Two observation sections in Case River and one online observation section in Case Tree were performed. Overall, we collected 65 transcript A4 pages and some other documents, such as meeting minutes, emails and observation notes as material for analysis. All transcripts were sent to interviewees for confirmation. The interview is semi-structured and consisted of mainly open-ended questions, hence allowing to follow and discuss interesting issues, or change the order of questions. The interview duration varied from 45 to 75 minutes.

The interview guide consisted of five parts: (1) background about projects, interviewee and his/ her participation in the project, (2) team collaboration challenges with different type of global boundaries, using the challenges from literature (Section 2.1) as an initial list, (3) reflection on collaboration infrastructure and artifacts as a coordination mechanism, (4) reflection on boundary spanner and boundary spanning activities as a coordination mechanism, (5) reflection on overall effectiveness of team coordination in the project and clarification on summary of the interview.

**Data analysis:**

We analyzed the collected data in two-stage analysis, following the recommendations by Miles and Huberman [41]. In first stage, we conducted a with-in case analysis, starting by
reading from each case all interview transcripts and relevant documents. The same data analysis framework was used for each case. The interesting text that expresses either challenges with organizational boundary, boundary spanner activity, boundary object and boundary leveraging practices were labeled with an appropriate code. Any text segment that links these concepts were coded as well. After that, we grouped relevant codes in to higher-order code using axial coding [33]. Transcript data was entered into the qualitative data analysis tool NVivo 8 for the ease of qualitative coding. At this point, no comparison between cases was made. In second stage, the concepts and their relationship were compared among four cases to identify commonalities and also variations among them, and to identify a number of best practices [40]. The overall process is summarized in Figure 2.

3.3. Case description

Table 1 describes characteristics of each project and the data collected from them.

a. Project River:
Project River was to develop and maintain a ship management system. The project is more than five years old and includes a large amount of legacy code. The project involves two external contractors, so-called Team Thames and Team Danube. Team Thames locates in Trondheim, Norway and implements the web application system. Team Danube locates in Stavanger, Norway and implements the desktop application system. The development methodology is Scrum. At the beginning of the project, there was little dependency between two teams as both teams only shared a common data access layer and database. At the time this study was conducted, both teams were performing a joint effort on refactoring the overlap code base, updating data model and developing a new module. Communication between two teams and with customer teams was performed mainly via phone and email. Site visits and workshop occurs sometimes during the project duration. We identified a team leader of Team Thames as a boundary spanner, who connected Team Thames’ members with customer and Team Danube.

b. Project Mountain:
Project Mountain was to develop a search engine system as part of a large IS management system. The search engine system has many API interfaces, which are used by many higher-level systems. The project is more than five years old, involving more than 1500 developers in a global scale. The part of the project that we investigated only relate to a team in the headquarter office in USA and two development centers in Norway, namely Team Everest, Team Dom and Team Eiger, correspondingly Therefore, there are high amount of technical, temporal and process dependency among these teams, and team coordination was performed as a daily activity. Team Everest developed part of the search function of a larger IS system, while Team Dom and Team Eiger were responsible for the core module of this search engine. The common set of coordination infrastructure and process were governed among all teams, including daily (virtual) meeting, teleconferencing meeting, email and TFS. Team Eiger and Dom have some local collaboration practices, such as communication via GIT, informal chat and frequent site visit. We identified a program manager in Team Eiger as a boundary spanner. He worked closely with other program manager from Team Dom, Team Everest and other teams, which form an informal integration group across locations.
Table 1: Case description

<table>
<thead>
<tr>
<th>Project type</th>
<th>River</th>
<th>Mountain</th>
<th>Ocean</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global arrangement</td>
<td>Inshore outsourcing</td>
<td>Insourcing</td>
<td>Offshore outsourcing</td>
<td>Open source</td>
</tr>
<tr>
<td>Project age</td>
<td>A customer team and two vendor teams in Norway</td>
<td>A development team in US and two development teams in Norway</td>
<td>A team in Vietnam and a client team in Japan</td>
<td>Network of contributors</td>
</tr>
<tr>
<td>Product</td>
<td>5 years</td>
<td>5 years</td>
<td>10 years</td>
<td>6 years</td>
</tr>
<tr>
<td>Team size</td>
<td>IS system</td>
<td>Search engine system</td>
<td>IS system</td>
<td>Network analyzer tool</td>
</tr>
<tr>
<td>Development method</td>
<td>Tailor Agile vs. Scrum</td>
<td>Scrum</td>
<td>Agile vs. Water fall</td>
<td>Unknown</td>
</tr>
<tr>
<td>Team structure</td>
<td>Hierarchy</td>
<td>Hierarchy</td>
<td>Hierarchy</td>
<td>Network based</td>
</tr>
<tr>
<td>Interview</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>notes Interview role</td>
<td>Project manager, team leader, developer</td>
<td>Program manager, developer</td>
<td>Bridge engineer, developer</td>
<td>Developers</td>
</tr>
</tbody>
</table>

**Project Ocean:**

Project Ocean was to develop a sub-system of a large-scale e-government solution. The project is more than six years old and the subsystem was outsourced to a development team in Vietnam in recent three years. The project involved a client team in Japan and several vendor teams in Asia Pacific area. In the scope of this work, we only consider team coordination among the client team and a vendor team in Vietnam, namely Team Baltic and Team Bering. Team Bering adopted Agile approach while Team Baltic used a tailored water fall development process. The tasks were sent to Team Bering iteratively and integrated to the main system by a joint team. The coordination practices between two teams are email, site visit and bridge engineer.

**Project Tree:**

Project Tree was to develop a network package analyzer. The project is open source, ongoing, ten-years-old and attracts a lot of participation from commercial firms. The development and product roadmap is done by a small group of active volunteer and commercial developers. Project main communication channels are a mailing list and an issue tracking system. The project also organizes annual conference to exchange experiences and practices among developers and users of the product. We studied the coordination between a commercial firm Y and the Project community. Firm Y developed a prototype and integrated modules from Project Tree, there is a need to understand technical insight and also keep up-to-date with changes in the project. We identified a gate keeper in Firm Y that is responsible for contributing to Project Tree via giving patches, finding and fixing bugs and participating in communication with other project participants.
4. Results

4.1. Capacities of boundary spanners

We identified four main capacity of a boundary spanner, regardless his role is formally assigned or emergent. The boundary spanner capacity includes the ability to attract team member recognition, holding multiple perspective expertise, decision making ability and work time flexibility.

**Team recognition**: in all cases, boundary spanners were recognized by not only internal team members, but also external stakeholder as an interface to communicate with the other side of the projects. For an internal team member, boundary spanners were seen as a closest position to external stakeholders and have influence on them: “I believe that the project owner trust us deeply. That is almost the problem. They trust so much that the way of working with us that let to [Team leader name] as an assumption part of technical insight. If [Team Thames] exploded and disappear the customer will have problem. I don’t think [Team Thames] is easy to replace at this time” (a developer from Team Thames, Project River). For a customer, boundary spanners were considered as a source of information and also a target for giving feedback and negotiating tasks (Project Mountain, Project Ocean).

**Multiple area expertise**: in project River, the team leader had knowledge of not only client side systems but also server side business logic and implementation framework: “[Team leader name] has worked with customer for so long so he knows about technical detail and domain logic much more than us” (a developer from Team Thames, Project River). In project Ocean, the bridge engineer hold knowledge about customer requirement, development framework and also expertise in testing: “One capacity of a bridge engineer is to be keen on the technology and the development framework, but also be fast on learning business domain and requirement from customers … I also checked the deliverables from [Team Behring] before checking into their system.” (a developer from Team Behring, Project Ocean). In Project Tree, a gatekeeper is keen on a general network protocol as well as the specific domain protocol that is locally used in FirmY.

**Decision making ability**: an important capacity of boundary spanner is the ability to make decisions. In project River and Mountain, decision making on how to leverage boundary distances and assign tasks were parts of spanner’s responsibility: “But when you run the projects and try to resolve the conflict and dependencies, that happening up here (management level). A lot of project tracking is done up here. For a big project if this one is late what consequences to the other team? I need to involve in all these stuffs” (a project manager from Team Eiger, Project Mountain). In Project Ocean, an bridge engineer was nominated some rights to make decisions in the onsite locations: “Sometimes, there are many small issues that you need to decide onsite, it will be unnecessary and too much bureaucracy for asking permission. So normally we are assumed to be a decision maker for these issues” (a developer from Team Behring, Project Ocean).

**Work time and place flexibility**: as a special position that stand across multiple boundaries, including temporal and cultural boundary, a spanner need to be flexible on working time and place. In Project Mountain, program manager often need to shift work time or work overtime to participate in meeting from other time zone: “… last night I have 4 meeting in a row from 9h30 am to midnight. We do it from home and all necessary tools are available…” Office is
empty at 5 o clock. People go home, have dinner and if we need to do some works we do it later…” (a project manager from Team Eiger, Project Mountain). In Project Ocean, a bridge engineer needs to stay 6 months in the customer sites for the first time and frequent visit after that: “A bridge engineer needs to be ready for a frequent onsite trips. Last year I stayed 6 months in the [Team Baltic]. This year, I have visited them 3 times, but in a shorter trips”. In Project Tree, the work time on the OSS project was mixed of office work time and personal time: “Initially, I participated in the [Project Tree] as my personal interest. As the company hired me, I started to promote [Project Tree] to the company. Nowadays, I work on the features needed by the company in my official work time. At home, I work on other things” (a gatekeeper from Project Tree).

4.2. Boundary spanners as team coordination mechanism

Boundary spanners perform four types of activities that facilitate resolving team interdependencies, which are mediating status information dependency, mediating task dependency, mediating practice flow, and leveraging global boundaries, as shown in Table 2.

Mediating status information dependency consists of two sub-concepts: transporting information and hiding information. The bridge engineer in Case Mountain and the program manager in Case Ocean communicated project status as one of the main tasks. Information about who do what, and current status of development tasks is formally exchanged in a daily meeting of the manager group (Case River) or weekly report to customer team (Case Ocean). In Case Tree, a gatekeeper subscribed for changes from the OSS community and informed management level about the status of the release.

In case River, the team leader was the only person in Team Thames was informed about Team Danube’s work status from the customer. This information was not frequently circulated in Team Thames, but when a problem arises, such as a bug or possible delay of a joint task, Team Thames members are informed by the team leader: “I need to be sure that everyone got what they needs, nothing more. We often have indirect contact with [Team Danube] via customers. We almost always send email with the copy to relevant product owners and vice versa. For some tasks, I am informed about [Team Danube] from the customer” (a team leader from Project River).

<table>
<thead>
<tr>
<th>Concept</th>
<th>Code</th>
<th>R</th>
<th>M</th>
<th>O</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mediating task dependency</td>
<td>Comprehending and transferring tasks</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Integrating deliverables</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Negotiating tasks</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mediating status information dependency</td>
<td>Transporting status information</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Hiding information</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Leveraging global boundaries</td>
<td>Translating linguistic information</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Shifting work time</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Site visit</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>Mediating practice flow</td>
<td>Facilitating exchanged practices</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Sharing knowledge and expertise</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
</tbody>
</table>

\(x\) represents for the presence, while \(-\) represents for the absence of the code

217
Interestingly, boundary spanners not only transport the status information but also filter some information to be shared. The information, such as detail development process (Case River, Case Ocean) and specific technical insights (Case Tree) were considered by boundary spanners when giving information about status of a task: “Normally the customer wants to know everything about what happens in the process. As we have some customized processes, we do not share all of information about how we develop the functionality. I was told to make decisions about negotiating these type of issues as well” (a bridge engineer from Team Bering, Project Ocean).

**Mediating task dependency** is the most frequently mentioned concept, which include comprehending and transferring tasks, negotiating tasks and integrating deliverables. There was often a discussion thread between a spanner and customers or other team (via formal meetings or exchanged emails) to clarify a new task linguistically and semantically (Case Ocean, Case River, Case Mountain). In Project Mountain, program manager also wrote requirement description of a new task and communicate request of changes from other teams. We also observed that boundary spanners also take care of dividing tasks for other team members: “I know nothing about them [Team Danube, Project River]. Actually I receive tasks from [the team leader name, Team Thames, Project River], he is not exactly a project manager but more like a project organizer. So he received the tasks from [Danube team] and split the tasks for us …” (a developer from Team Thames, Project River).

Moreover, the scope and schedule of a task were negotiated by boundary spanners between vendor teams (Case River) or between a customer team and a vendor team (Case Ocean, Case Mountain). Negotiating often required prior experience and knowledge about similar tasks: “I need to estimate a feasible duration for completing the task. This is often done based on my previous experience. If the customer give too short period for a tasks, I need to report to the project manager and negotiate this if necessary” (a bridge engineer from Team Bering, Project Ocean). In Project Tree, the gatekeeper had done some discussion to influence the development of a feature to be aligned with his company’s requirement.

In Project River, Ocean and Tree, boundary spanners also involved in integrating and validating source codes from team members before delivering to customers or other vendor team. A developer from Team Thames said: “[A new task] will be coordinated with [Team Danube, Project River]. The project leader is doing some merge tasks today and after he is done with those, I will merge my changes to the main branch and he will merge all to the development branch”. The integration was done by allowing checking into boundary spanner’s development branch (Case River) or by sending patches (Case Ocean, Case Tree, Case Mountain): “[Project Tree] is a very large project with many components so it really depends on the specific task to determine the appropriate route. …The [Project Tree] package maintainer usually has the task of being the gatekeeper for those bugs that have been reported against [Company name] products by the customers or the support teams, and he may channel them to the upstream Bugzilla if applicable” (a gatekeeper from Project Tree).

**Leveraging global boundaries**, such as geographical boundary, temporal boundary and cultural boundary is also a part of boundary spanner’s activities. We observed that site visit was commonly adopted in Case River, Mountain and Ocean to reduce negatives effect of geographical boundary. Shifting work time or even going onshore was done in a project with small overlap time window (Case Mountain, Case Ocean): “… last night I have 4 meeting in a row from 9h30 am to midnight. We do it from home and all necessary tools are available…”
Table 3: Boundary objects adoption

<table>
<thead>
<tr>
<th>Category</th>
<th>Object</th>
<th>R</th>
<th>M</th>
<th>O</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing common infrastructure</td>
<td>Collaboration tool</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Providing standard</td>
<td>Code guideline</td>
<td>+</td>
<td>+</td>
<td>n/a</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Code/test generation</td>
<td>+</td>
<td>-</td>
<td>n/a</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Request of Change form</td>
<td>n/a</td>
<td>+</td>
<td>n/a</td>
<td>+</td>
</tr>
<tr>
<td>Providing transportation device</td>
<td>Set of product feature</td>
<td>n/a</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Desktop screen</td>
<td>+</td>
<td>n/a</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Paper prototype of</td>
<td>+</td>
<td>n/a</td>
<td>+</td>
<td>n/a</td>
</tr>
<tr>
<td>Triggers actual coordination</td>
<td>A software patch</td>
<td>n/a</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Piece of code</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>+</td>
</tr>
</tbody>
</table>

+ represents for the positive effect, represents for the negative effect and n/a is not applied in the case

Office is empty at 5 o’clock. People go home, have dinner and if we need to do some works we do it later…” (a project manager from Team Eiger, Project Mountain). In case Ocean, a bridge engineer was also a translator of requirement documents for local teams: “…one of my task is to translate requirement documents into a friendly and localized document for the development team at home…” (a bridge engineer from Team Bering, Project Ocean).

Mediating practice flow is the less mentioned concept from data transcript. In Project Mountain, the gatekeeper facilitated the exchange of development practices and sharing knowledge expertise as part of organizational learning process: “We have a policy that people should visit [Team Everest] once per year. The purpose is to get to know people, to make the communication smoother for distance collaboration. If you have met a person in the same room and done some work with him it is much easier to pick up the phone later. And also to understand the other site how they are work. Seating far away you may ask why they are doing that? You don’t know how the team organized.” (a program manager from Team Eiger, Project Mountain). In Project Tree, some good development practices and knowledge about how to configure the network protocol were discussed in the annual workshop and reported to Firm Y by the gatekeeper.

Figure 3: Conceptual framework on boundary spanning as team coordination mechanisms
4.3. Boundary object as a coordination mechanism

We observed a set of artifacts that are used by both sites of the projects and classified into four types: infrastructure object (Team Foundation Server, Issue Tracking System, Mailing list), object for standard (code standard, code and test generation tool, request of change), transporting object (paper prototype of business flow, product feature list, desktop screen) and coordination trigger (software patch and piece of source code), as given in Table 3.

Providing common infrastructure for team collaboration was the initial goal for using infrastructure objects. In Project Tree, issue tracking system served as a primary channel for coordinating tasks. In Project Ocean, the collaborative development environment was adopted in Team Behring, with advanced features such as virtual meeting, desktop sharing and joint task definition. In Project River, the collaboration tools were not adopted in the same level in both sides, as mentioned by a developer from Team Thames, Project River: “[Collaboration tool name] has a lot of features which are too much bureaucracy for our way of working. We work in a very informal agile and lean manner. We don’t spend time on talking about things that don’t need to talk about and working on structuring issues”. However, Team Danube adopted most of collaboration supported feature of the tool and also use it in communicating customers: “It [Collaboration tool name] solves specific development projects because it highlights stuff that developers need to know. If some you break the build that another developers depends on, you get the information right out (a developer from Team Danube, Project River). Consequently, the collaboration tool was not actually used as a communication tool between two teams in Case River, due to the asymmetry on adopting the collaboration tool between both teams.

Providing standard to support boundary-spanning activities and enable coordination beyond the team and project boundaries was extracted from interview transcripts in all cases. Standard objects were hard to change and be part of a formal development process. For Project River and Mountain, code guideline was an artifact for establishing a common understanding of code structure and coding style, and a basic for discussing the quality of code among sites: “We haven’t established guideline for creating new things and refactoring ‘boiler plate’ code. But toward developing shared functionality on higher level than the data access layer, which could be used in both our applications, we agreed on having very high standard, such as 100% test coverage of the code, and establish code guideline. If all developers don’t comment or clean over it you experience kind of low quality” (a developer from Team Thames, Project River).

A developer from Team Danube of Project River also said: “If you establish a guideline for coding, you have a lot of tools you can achieve the quality you want in the project. And it should not depend on whether you seat on other country or city or whatever” In project Tree, code guideline was available both in Firm Y and Project Tree. The gatekeeper in Project Tree mentioned that the code guideline in the OSS project was less strict than the code standard of Firm Y, making it easier to adopt for a heterogeneous group of participants. Code/ test generation tool were effectively used in three cases. In case River, code generation was used to automate some functionality development by both Team Thames and Team Danube. In case Tree, test generation was the only testing activity is done in common in the projects. Therefore, the test parameters and test result served as a boundary artifact among developers and testers. Request of Change Form was a part of a formal process when a developer or a program manager wanted to change some parts of source code beyond the team boundary.
Providing transportation device: Paper prototype of business flow, set of product feature and desktop screen provided a common textual and visual representation of (part of) the product. The transportation objects were the target to discuss and flexible to change. A developer from Project River said: “We tried to communicate things which touch up on the business domain but sometimes it is not apparent at the time that you touch the other person’s business domain and you could potentially and actually break something in their application from fixing something in our application”. Another developer mentioned: “… in some case we shared desktop screen to explain the complex design issues or a hard-to-replicated bug”. We also observed during the workshop section, a large poster of an overall flow diagram of the new functionality and a list of paper showing GUI design for both teams were used (Figure 4).

Triggering actual coordination is done by discussing directly on products, such as software patches or piece of source code. In project Mountain, Ocean and Tree, the patch was either checked in to the main branch or sent to the relevant stakeholder for review and integration: “We don’t often create branches. We make something call [A software patch name], which is kind of data we develop and send to others who want to test it or integrate it into their system. Every bug is really a [A software patch name]. When we check in we can use the [A software patch name] to see what has been checked in so we can take it and put in other branches” (program manager from Team Eiger, Project Mountain). In project Tree, software patch included a self-description of the patch and relevant context information: “Developers does not shared source code directly but communicating a software patch, which includes a self-description and proper commented codes” (observation from Project Tree).

5. Discussion
5.1. Coordination effectiveness in four cases

The four projects are reasonable representative of different organizationally distributed projects and coordination needs. In Project River, communication across team was mainly done via the boundary spanner. Infrastructure object was shared but not effectively used to coordinate developer tasks due to the misalignment of governance policy and lack of commitment on using the tools. Therefore, most of the time, developers from both sites did not know about other team’s status. Boundary object such as desktop screen, paper prototype, code guideline was used mainly to support the cross-boundary activity of the team leader and other developers. The case shows that even in small team with low task uncertainty, coordination challenges still occurs.
In Project Mountain, a group of program manager acted as a integration team to facilitate cross-team coordination. A set of collaboration infrastructure was applied in all locations. The adoption of boundary object was governed by a global process and complement for boundary spanner’s activities. Therefore, it made an impression that the developers get an effective support to understand a big picture of who-do-what. Compared to Project River, the case showed that even in a highly geographically distributed project, coordination problem can be mitigated by aligning development process, appropriate practices and boundary objects. The main coordination issue is to cope with time zone difference and limited face-to-face meeting.

In Project Ocean, boundary spanner was also the primary collaboration channel. Due to the large difference on practices between vendor and customer organization, the collaboration tool did not give much help. Team Behring relied heavily on the bridge engineer as more than 50% of his task devotes for team communication. The uncertainty of tasks from customer increased the passive activities of boundary spanner. In this case, boundary objects acted as information back-up for boundary spanners. The combination of boundary spanners and objects helped to reduce risks for knowledge management process to extract and maintain tacit knowledge of the projects in the organization, and threats of information bottom neck in the social network.

In Project Tree, boundary objects, including both infrastructure objects, transport objects and trigger objects substituted for other coordination mechanisms. As participants are assumed to have the same interest when participating in the project, the variety in knowledge and practices are reduced. The self-explained software patch and issue description form enabled coordination without communication. As a result, there was little problem regards to team coordination in Project Tree.

5.2. Refined framework on boundary spanning as team coordination

To response to RQ1, we observed four common capacities of boundary spanners in all cases, namely recognition by stakeholders, multiple area expertise, decision making ability and work time and place flexibility. As boundary spanner-in-practice have to develop both an inclination and ability to be a legitimate peripheral participant in both groups being spanned [19], the team recognition and multiple area expertise is a result of these peripheral activities. We also agreed with previous studies about the decision-making and leadership as part of core capabilities of a boundary spanner [42]. Cataldo et al. observed that the formalized boundary spanners tended to contribute less to the development effort than the rest of the developer, while informal boundary spanners emerges from a highly productive developer [43]. Complement for this finding, we showed that informal boundary spanners contribution might not be properly recognized by the management level.

To response to RQ2, we found that boundary spanners influence team coordination by mediating status information dependency, mediating task dependency, leveraging global boundaries and mediating practice flow. Prior to this work, we can find only two studies on boundary spanning as team coordination mechanism in software projects [21, 25]. Strode et al. looked at boundary spanning as a coordination strategy in four co-located projects [21]. Compared to their conceptual framework, we showed that boundary spanning not only support explicit coordination (right place, right thing, right time [21]) but also facilitate the exchange of know-who, know-what and know-how in the team. Besides, we highlighted the boundary leveraging function of boundary spanners in distributed context, which were not
visible from collocated projects [21]. Krishnan et al. examined the usefulness of boundary spanning in collaborating distributed team via project social capital perspective [25]. Compared to their conceptual model (main constructs are spanning knowledge boundary, social boundary and operational boundary), we identified that spanning practice boundary also take a role in influencing team coordination. While Krishman’s model based on literature, our framework is grounded from empirical data in four different types of global software projects.

To response to RQ3, we found that four types of boundary objects, which are common infrastructure, standard, transportation device and coordination trigger can complement, substitute and back up for boundary spanners. Our findings contribute to the current understanding of the use of boundary object in the context of software development. The initial framework on the role of boundary spanning in team coordination is further refined by qualitative findings, as shown in Figure 3. It illustrates how boundary spanner activities and the use of boundary object to leverage team interdependency across organizational boundary.

5.3. Threats to validity

While designing and conducting this study, various conscious decisions were taken to strengthen the validity of results. Using the checklist proposed by [37] we evaluated the case study design. This ensured that we had addressed all the critical requirements of case study design including aims of the study, defining the case, unit of analysis and data collection methods among others.

Construct validity refers to the relations between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. In this study, the interview guide was generated from theory and previous literature, which reduce this validity threat.

Internal validity refers to the insurance of casual relationships between treatment and outcomes. In this study, the role of boundary spanning were examined in the context of global software projects, and the its’ relation with other team coordination mechanisms was discussed. However, given the complexity of a real-life case and the fact that confounding factors always presents when studying real world systems, the isolated effect of boundary spanning cannot be established, and requires further investigation.

External validity concerns about the generalization of observed results to a larger population. In this study, four cases represents various type of inter-organizational software development projects in project size, global arrangement and product domains. The typical and extreme case selection strategies reduce this validity threat.

Credibility concerns about the confidence of the result. We adopted several threat mitigation strategies, such as sending interview transcripts to each interviewee for correction; involving three researchers on study design and review, and detailed documented steps of data collection, processing and analysis. Thus we believe that the resulting detailed protocol serves as a good means to replicate this study. Last but not least, we also used multiple data sources and multiple viewpoints are used as data triangulation to strengthen the evidence generated in the case studies.
6. Conclusions

Many research efforts has been focused on spanning activities across cultural boundary in IS and SE literature. In this study, we made contributions to current body of knowledge on team coordination in GSD by (1) characterizing the capacity of boundary spanners as a project coordinator, (2) identifying the boundary spanning activities to address global boundaries and (3) illustrating the role of boundary object in supporting team coordination.

Our study has several implications for practitioners, i.e. project managers and team coordinator who would like to improve team coordination effectiveness. Firstly, boundary spanner and boundary objects are both important coordination mechanism. In a context where one of them are inhibited or ineffectively adopted, the better focus on the other mechanism can help to compensate. Secondly, our cases reveal that the availability of boundary objects is not sufficient to obtain the full use of them. A governance policy that goes beyond organizational boundary should be adopted to establish the consistent adoption of them. Thirdly, while boundary spanners are important for team coordination, they also have some issues, such as knowledge silos, role conflicts and stress. Therefore, the pros and cons of adopting boundary spanning activities should be taken into account. Their contribution should be recognizing and aligning boundary spanner’s goal with organization’s goal.

Future work can extend this study by (1) perform a longitudinal study on addressing questions about “when boundary spanner leverage boundaries” or “when boundary object is effectively used?”; (2) elaborating our conceptual model and derive measurable hypotheses and (3) identifying how and what factors drive the decision-making procedure of boundary spanner when mediating task and information flow.

Acknowledgment

We would like to thanks all companies who participated in this research provided us with valuable information about their work and activities.

References


[18] E. Hochmüller, “The requirements engineer as a liaison officer in agile software development”, Workshop on Agile Requirements Engineering, Lancaster, United Kingdom, 2011


225


Paper MP7 - Forking and coordination in multi-platform development: a case study

Forking and coordination in multi-platform development: a case study

Anh Nguyen Duc¹ ², Audris Mockus², Randy Hackbarth², John Palframan²
¹ Norwegian University of Science and Technology, Sem Saelands vei 7-9, Trondheim, Norway
² Avaya Labs Research, 211 Mt Airy Rd, Basking Ridge, NJ, USA
anhn@idi.ntnu.no,{audris,randyh,palframan}@avaya.com

ABSTRACT

With the proliferation of desktop and mobile platforms the development and maintenance of identical or similar applications on multiple platforms is urgently needed. We study a software product deployed to more than 25 software/hardware combinations over 10 years to understand multi-platform development practices. We use semi-structured interviews, project wikis, VCSs and issue tracking systems to understand and quantify these practices. We find the projects using MR cloning, MR review meeting, cross platform coordinator’s role as three primary means of coordination. We find that forking code temporarily relieves the coordination needs and is driven by divergent schedule, market needs, and organizational policy. Based on our qualitative findings we propose quantitative measures of coordination, redundant work, and parallel development. A model of coordination intensity suggests that it is related to the amount of parallel and redundant work. We hope that this work will provide a basis for quantitative understanding of issues faced in multi-platform software development.

Categories and Subject Descriptors: D.2.9 [Software Engineering]: Management General Terms Management, Measurement, Human Factors

Keywords: Multiple platform development, fork, coordination, empirical study

1. INTRODUCTION

A rapid growth in the popularity of cross-platform software systems has been brought on by the recent proliferation of mobile applications. Software such as web browsers, instant messaging clients, voice communication, e.g., Skype and other mobile applications, needs to be delivered for an ever increasing range of hardware and software platforms. Skype, with 299 million users, is available on Windows, Linux, Mac and Android [27]. The market for cross-platform mobile development tools is forecast to reach US$8.2 billion by 2016 [2].

The existing literature on multi-platform development is highly pragmatic, with guidelines and technical solutions [11, 1, 28, 22], but with few empirical studies of software development practices and coordination needs that would support multi-platform development. The traditional software engineering approaches may not directly apply in a multiplatform application context that has many unique challenges. On one hand, development teams have to conform to different requirements and constraints for each platform (e.g., iOS, Android, Windows 7, etc.), hardware (e.g., HTC, Google, Samsung), and, potentially, communication protocol (e.g. AIM, XMPP, SIP, H323). On the other hand, developers need to maintain inter-operability, consistent performance and, potentially, look-and-feel of the product across different platforms [9].
The maintenance and upgrades, which account for two thirds of a typical software project’s lifetime cost [23], might take even more effort in a cross-platform system development. Not many developers are intimately familiar with several platforms, thus it is often necessary to have different individuals and teams implement the same product for each platform. As a result, it may be difficult to propagate bug fixes across multiple version of a software application. For example, a recent critical security flaw was fixed for iOS, but remained unpatched for Mac OS [10]. Some code, often the user interface, has to be implemented in a platform-specific framework and ensured that the application features work in a similar fashion. This requires the teams implementing on different platforms to be able to effectively coordinate their work.

To understand how software intensive organizations resolve development and coordination problems in multi-platform development we set out to study a set of voice, video, and instant messaging clients at Avaya. The qualitative part of the study investigates the state-of-the-practice in multi-platform development and associated coordination mechanisms. In the quantitative part we constructed a network of related file versions across 43 instances of version control systems used by 25 different software and hardware platforms and use it to quantify the extent of forking and coordination. Our primary contributions involve:

1. A description of industrial multi-platform development effort: the practices and challenges; coordination mechanisms; and reasons for codebase forks.
2. We propose multi-platform development measures for the extent of parallel development, coordination, and redundant work.
3. We investigate the relationship between the intensity of coordination and the amount of parallel development and redundant work.

The paper is organized as follows: Section 2 introduces related work on cross platform development, team coordination and branching practices. Section 3 presents research approaches and study context. Section 4 presents findings from qualitative phase. Section 5 describes measures of forking and coordination, Section 6 presents quantitative findings. Section 7 reviews related work, Section 7.3 points out limitations, and Section 8 presents the conclusions.

2. TERMINOLOGY

We start from the terminology used in this paper:

- Multiplatform software refers to the ability of software to operate on more than one platform with identical (or similar) functionality [6]. Platform can refer to (1) different types of operating systems (e.g., Windows, FreeBSD, Linux, Mac OS X, and Solaris systems), (2) types of processors (e.g., x86, PowerPC, SPARC or Alpha), (3) communication protocol (e.g. SIP, RTM, AIM and H323) and (4) types of hardware systems (e.g., iphone, ipad, android phone and windows phone).

- Project has its own code repositories with active code commits and MRs and results in one or more releases of a client product. In this study, project, platform and fork are used as inter-change terms.

- Fork is a clone of an entire VCS repository that has subsequent development
independent of the original project [20]. Modification request (MR) is a piece of work (a story, an epic, a task or a bug) that is tracked in issue tracking system, such as JIRA. Commit is a set of changes to one or more file traced by Version Control Systems (VCSs). Commits to the VCS are associated with a MR identifier in the considered projects.

- Software dependencies refer to the relationship between software artifacts, such as source files, components and modules. There are many type of dependencies, such as data-related dependencies (i.e. a data structure modified by a function and used in another function) [12, 3] or functional dependencies (e.g. method A call method B) [12] and logical dependencies (e.g. files being changed together) [8].

- Related files are files that had identical content in the past (based on their version history) [17].

- MR-clone is an MR which was created for another project and is associated with an MR in the original projects. MR clones are used as a coordination mechanisms to track tasks among different projects.

- Coordination mechanisms are approaches to manage dependencies among software artifacts and project stakeholders, and is an important research topic [15].

3. RESEARCH METHOD

We describe the context of the study in 3.1 and design in Section 3.2.

3.1 Context

Avaya is developing large, complex, real-time software systems that are embedded and standalone products. Development and testing are spread through 10 to 13 time zones in the North America, USA, Europe and Asia. R&D department employed many virtual collaboration tools such as JIRA, Git, WIKIs and Crucible. Development teams use Scrum-like development methodologies with a typical 4-week sprint. Several mechanistic coordination approaches have been used, such as (1) daily scrum meeting, (2) sprint planning meeting, (3) sprint review, (4) backlog review meeting, and (5) Scrum of scrums meeting [24].

We focused on a 10+ year old software component, the so-called Spark engine. As a software platform, Spark provides a consistent set of signaling platform functionalities to a variety of Avaya telephone product applications, including those of third parties. The current evolution of Spark is a client platform that provides signaling manager, session manager, media manager, audio manager and video manager. Spark engine started in 2005 and has forked into many different codebases to support different combinations of hardware (Flare, ipad, iphone, Mac, Samsung devices, 1XC, VDI, ACA, AAC, IPO and LYNC), software (iOS, android, windows) and communication protocol (SIP, H323, etc), as shown in Figure 1.

3.2 Research design

We conducted a single case study in a large telecommunication organization, following a guideline suggested by Yin [29]. The case represents a typical multi-platform software development with involvement of teams across geographical and organizational boundary.
Figure 1: Research phases

Exploratory case studies are used to investigate little known phenomenon or one without an established theoretical basis [21]. We use this approach to discover the code divergence, forking activities and coordination mechanisms in multi-platform software systems.

The research consists of two phases and was conducted between Dec-2012 and March-2014. In Phase 1, we aimed to understand the context of multi-platform development, including technical and management challenges, and current development and coordination practices. A range of data collection methods was used to achieve familiarity with the organizational context and participants, including conversations and interviews, and review of relevant documentation and project artifacts. Semi structured interviews is the primary data source in this phase. We conducted 10 semi-structured interviews with different project roles, such as product manager, technical leader, developer and researcher, as shown in Table 3. The interview guide consists of four main parts:

1. Challenges of working on inter-platform teams,
2. State-of-the-art of codebase divergence,
3. Coordination mechanisms, and
4. Codebase merging activities

After all data was extracted, we performed data cleaning to remove inactive repositories (repositories without changes in the last 12 months) and moved repositories (repositories moved from one VCS to another VCS without subsequent development activities in the original VCS). 20 inactive repositories and 4 moved repositories were removed. Table 2 provides a summary of the final dataset of 43 project repositories for the 25 platforms.

Table 1: Summary of dataset

<table>
<thead>
<tr>
<th>Items</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of platforms</td>
<td>25</td>
</tr>
<tr>
<td>Number of codebases</td>
<td>43</td>
</tr>
<tr>
<td>Number of files</td>
<td>232065</td>
</tr>
<tr>
<td>Number of branches</td>
<td>571</td>
</tr>
<tr>
<td>Number of commits</td>
<td>4055968</td>
</tr>
<tr>
<td>The timeframe</td>
<td>2010-2013</td>
</tr>
<tr>
<td>ID</td>
<td>Positions</td>
</tr>
<tr>
<td>----</td>
<td>----------------------</td>
</tr>
<tr>
<td>P1</td>
<td>Scrum Master</td>
</tr>
<tr>
<td>P2</td>
<td>Developer</td>
</tr>
<tr>
<td>P3</td>
<td>Developer</td>
</tr>
<tr>
<td>P4</td>
<td>Development manager</td>
</tr>
<tr>
<td>P5</td>
<td>Developer</td>
</tr>
<tr>
<td>P6</td>
<td>Development manager</td>
</tr>
<tr>
<td>P7</td>
<td>Scrum Master</td>
</tr>
<tr>
<td>P8</td>
<td>Product owner</td>
</tr>
<tr>
<td>P9</td>
<td>Developer</td>
</tr>
<tr>
<td>P10</td>
<td>Development lead</td>
</tr>
</tbody>
</table>

Based on findings in Phase 1, we proposed and described several measures of parallel development and coordination. We performed logistic regression analysis to investigated the relationships among some of the concepts.

4. QUALITATIVE FINDINGS

In this section we summarize our qualitative findings organized into three themes as shown in Figure 2:

- Challenges managing diverged codebases (Section 4.1),
- Reasons for diverged code bases (Section 4.2), and
- Coordination mechanisms deployed to manage diverged code bases (Section 4.3).

4.1 Challenges managing diverged codebases

We found four primary challenges in managing diverged code bases.

- Technical debt is the extra development that arises when code is implemented in the short run (e.g. to deliver a product release) in a manner that will require additional maintenance and development in the long run (e.g. to deliver subsequent releases or to share code among similar products). Technical debt is often necessary, but we found that a diverged code base encourages the growth of technical debt as described in Section 4.1.1.
- A diverged code base encourages redundant development of features as described in Section 4.1.2.
- In addition to redundant development a diverged code base encourages redundant test effort as described in Section 4.1.3.
- Developers and testers who work on a diverged code base tend to become experts on their fork of the code base, but not on the family of products from which the code base originated as described in Section 4.1.4.

The interview guide was designed using the checklist from the literature review and revised multiple times by the authors. Each interview lasted from 30 to 60 minutes, depending on
time availability of interviewee. The interviews were flexibly conducted to allow for improvisation and exploration of emerging issues [21]. The interviews were recorded, transcribed and reviewed by interview participants.

Relevant segments of text that expressed either challenges and practices in multiple development platforms were labeled with an appropriate code by open coding. After that, we grouped the relevant codes as a higher-order code using axial coding [4]. Finally, we performed selective coding in secondary data materials, such as project wiki and MR descriptions.

In Phase 2, we aimed at quantifying important concepts found during Phase 1, as shown in Figure 2. In particular, we quantified the extent of forking activities, amount of coordination via cloning MRs and the extent of potentially redundant development. The primary data sources in this phase were Version control systems (Git and SVN) and Issue tracking system (JIRA). We started from data integrated over all Avaya projects as described in previous studies [16, 17]. The data unifies information from most of version control and issue tracking systems in Avaya, but we still need to select the projects related to the communications client’s. Three authors of the papers had insights on the products, which helped to identify 67 relevant codebases. We interviewed key stakeholders to confirm and complement the codebase list. To select and link relevant data, we wrote shell, Perl, and R programs to extract and calculate the necessary measures.

4.1.1 Growing technical debt

We found that the multi-platform development increases the technical debt for the whole product family. P10 mentioned: “We migrate software from platform to platform, or team to team, and this builds up technical debt. We need to continuously invest in software — it is getting old even if you don’t touch it”. Spark diverged codebase has many individual product teams. Because of overall staff constraints, each product team is thinly staffed and yet is responsible for the entire product. As a result, schedules are tight with little time to reduce the MR backlog: “Chasing a deadline created technical debt. Technical debt needs to be caught and dealt with earlier.” (P8).

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrA</td>
<td>Engine components in phone B189 using protocol H323</td>
</tr>
<tr>
<td>PrB</td>
<td>Engine components in series phone 96x1 using protocol H323</td>
</tr>
<tr>
<td>PrC</td>
<td>Engine components for Flare in iOS</td>
</tr>
<tr>
<td>PrD</td>
<td>Engine components for 1X apps in mobile devices</td>
</tr>
<tr>
<td>PrE</td>
<td>Engine components for Flare in Windows</td>
</tr>
<tr>
<td>PrF</td>
<td>Engine components for Flare in Android</td>
</tr>
<tr>
<td>PrG</td>
<td>Engine components for phone apps using protocol SIP in iOS</td>
</tr>
<tr>
<td>PrH</td>
<td>Core engine functionalities for a set of client products</td>
</tr>
<tr>
<td>PrI</td>
<td>Engine components for Flare in iPad</td>
</tr>
<tr>
<td>PrF</td>
<td>Set of functionalities for thick desktop oneX endpoint clients</td>
</tr>
</tbody>
</table>
In several projects, multi-platform development comes with rapidly increasing MR backlog, creating pressure on developers, which leads to “work around” MR resolutions. Here is an example: “The project has an ever growing backlog … he development team is stretched thin. We are focusing on getting [release] out, so there is no bandwidth to tackle the backlog. Many issues are open though internal Interop, Product Verification (PV), or System Verification (SV) testing. Many of these are duplicates. If it were not for the urgency of shipping, the backlog would be addressed. We have stability issues we do not know the cause of …” (P5). A short-term resolution for a bug or a feature request that is applied to only a subset of platforms would need more work later to be applied to the remaining platforms.

4.1.2 Redundant development effort

We found that product team staff responsible for one Spark platform were often unaware of many of the changes in other Spark platforms. As a result, features and bug fixes are implemented independently for each platform resulting in development that is unintentionally duplicated. Here is an example: “Some files included in the roofs are present multiple times in different codebases. For example some files are generated by multiple packages. This is redundant but also makes it difficult to know for sure which version of the file ends up on which phone release. … it is important to make sure that you identify the correct source for the file as found on the latest phone release today and remove the other ones. It is more difficult to check across different platforms.” (P3)

Occurrence of multiple identical slightly dirveged files complicates merging features across platforms. “Team PrA will have to do much of the duel delivery. Many features delivered to PrA need a port to PrB too. But the Team PrB will be involved in the code reviews. They will need to get involved in case the merge does not go smoothly.” (P2)
In particular, the replication of code related to third parties components are more critical and harder to solve. P1 talked about collaboration between PrA and PrB: “We found a bug in PrB codebase around the Broadcom hardware. Broadcom provided a fix ... and also introduced patches for that. We were working with Broadcom on PrA and would propagate the fix to PrB. ... We were not in the position to take the patch in the current release of PrB. This happened a couple of times and then we had different patches from Broadcom on PrA and PrB. The main challenge in consolidating two branches was matching the Broadcom patches”.

To mitigate redundant development, some individual product teams on different Spark platforms have made an effort to track changes in the other team’s product as illustrated in this example: “... Product team PrA team needs to watch for every submission to Product team PrB. Product team PrA assigned a developer dedicated to the product of PrB, and have an established line of communication with them via phone and conference call throughout the week . . .” (P2).

4.1.3 Redundant testing effort

Diverged code bases encourage redundant test effort because each platform’s test team typically independently tests similar features or functionalities from different platforms. Unless they know other teams’ test cases of similar features, testers independently create their own test cases as shown in this example: “the PrA team now only tests for PrA code and PrB team only tests PrB code. If there is a cross delivery, both teams have to test other product as well. Hence, both test teams get hit by the same amount.” (P2)

We found that duplicated testing effort is more critical in high-level testing teams, such as at PV team and SV team. At this level, testers did not have insight about releases and how they are related across platforms. P8: “This is something outside of our hand. What we can do is to tell the SV team that these products should be tested, there is a new release with these patches, we need you to pick it up and run with it. . . We do that by writing in the release note. . . . But usually the SV team organizes themselves to secure resources for program vs. testing these patches just like we do in development.”

A particular and often very important case of redundant testing is testing third party components: “when a bug involves both [third party vendor] hardware and application products, both sides need to be able to reproduce the bug to find the main root-cause.” (P6)

4.1.4 Lack of knowledge of the whole system

With diverged code bases, each project has its own priority of features and bug fixes. As a result of continuous evolution, the whole product family becomes extremely complex. Consider, for example, the likely complexity of the diverged code base leads to few, if any, engineers who understand in detail this entire code base.

In addition, individual development teams typically do not have “who does what” and “who knows what” knowledge about other teams as shown with this example: “Once you move outside the core, you have different platforms, schedules, and product managers, etc. In many cases working with another project is like working with a different company. There are different things to manage, so the orchestration of it all becomes problematic.” (P5).
4.2 Reasons for forking a codebase

In projects related to Spark we identified four reasons for creating forks leading to diverged code:

- Schedule constraints (Section 4.2.1)
- Technical variation in the platform (Section 4.2.2),
- Coordination overhead in maintaining a single platform (Section 4.2.3), and
- Organizational differences (Section 4.2.4)

4.2.1 Schedule-driven forks

We found that a common reason for code divergence is different project release schedules. In particular, one project may require a stable code base with minimal, tightly controlled changes while another project may be in active development with rapid code changes. For example, Project A shared some common features with project 96x1, and originally they were in the same codebase. However, Project A team created a separate code to assure a stable code base: “the next delivery date of PrA is [Date X] while the coming software delivery date of PrB is later at [Date Y]. Therefore, while PrB is still in development phases with testing and debugging, PrA needs to be stable for delivery”. (P1).

Projects with the same release schedule, but different development schedule have also chosen to diverge their code base. For example, we observed that code divergence occurred among projects with different numbers of sprints or with different sprint durations.

4.2.2 Feature-variety-driven forks

We have observed code divergence when one project introduces significant new features compared to the project(s) using the shared code base. For example, two projects PrE and PrF implemented a set of signaling engine functionalities in iOS and other mobile platforms accordingly. Early in its project stage, PrF stopped sharing the signaling engine codebase: “after the first six months, the signaling engine stopped synchronization with project PrE. From that time, it was branched off from PrE and developed independently, due to technical differences. The current situation is that it has features that other iOS clients do not use” (P5).

Another typical example of forking due to feature variety is project PrG and project PrD implementing client in Windows and iOS platforms. In the beginning, there was a close collaboration between the two teams as user interface code was shared for the first two releases: “… the reason is that the UI team is the same for both Windows and iOS versions. The design and logic behind them is pretty similar…” (P8). The code base for subsequent releases diverged, however, because of project differences in user interface requirements. Developers in PrG stated that this issue made it difficult to collaborate among teams.

The assessment of feature variety becomes a part of the forking decision-making process in PrA: “… the decision is usually made by trying to understand what are the main changes coming in from other releases. For example, if we are working for a H.323 release and we know that a release from another project introduces a major feature then probably we want to branch out before that. This is a purely risk management activity.” (P9)
4.2.3 Coordination overload

When two projects share the same code base, cross-project coordination is required to manage the impact of changes to the code base on each project. We have observed that projects have chosen to fork their code base when too much coordination is required. P6 stated: “... in general, there is no good reason to have separate platforms of the same hardware. Historically there were separate teams working on codebase PrF. Later in the project, they had been communicating too much and (hence) the codebases got separated ”. Here is another example: “... the reason for stopping synchronization (between PrC and PrD) is too much overhead, no dedicated people to maintain the [shared] source code” (P8). Note that according to our definition, we would refer to feature forks as branches.

4.2.4 Forking due to organizational differences

We have observed code divergence between two projects with common features and common schedule because the projects are in different organizations. The reason is that management needs (or wants) total control of resources if they are responsible for product delivery in their organization: “...it (coordination efforts) has been mostly on ipad and windows team, purely because they came from the same team right, because we were working very closely together on the same area . . . after that, we have been divided into different teams, as the managers were divided into the pure desktop and windows team and the purely mobile teams ...the organization split in the different ways from ...”

In Spark, codebases are forked for outsource development teams. It is a common case that the outsourced team created its own version of the code base to establish code ownership of features and bug fixes: “...I find the best thing working with external teams, especially from different timezone and locations is that if you have a really good clear process, especially with feature forking model, it is very easy for people to work together...they work on features and wrap it up, deliver back to mainstream in the end of release ...” (P7). Strictly speaking, we consider feature work to be an instance of branching, not of forking. However, in this case the outsourced team uses their own version control repository and work on an entire sequence of releases, not on just on individual features.

4.3 Coordination mechanisms across platforms

We observed three primary techniques for coordinating development among platforms with diverged code. Many projects deployed at least two of these approaches:

- Clone MRs across platforms (Section 4.3.1) where a cloned MR is a replica of the original MR, containing the same information stored in the original issue, e.g. Summary, Affects Versions, Components, etc. Cloned MR is typically linked to the original MR.
- Establish a cross platform coordinator role (Section 4.3.2)
- Conduct cross-project MR review meetings.

4.3.1 Cloning MR tickets across platforms

Formal defect tracking systems are used as a standard practice in all Spark-related development teams. To coordinate a common bug fix among projects, many projects create a cloned MR as described by P7: “…there is a platform MR ticket linked to an application MR
ticket that requires resolution, the ticket in the application would be assigned to a developer who is responsible to track the changes. He would do so through the relevant focal point in our team to the platform team. Once the platform ticket is resolved, the person assigned should verify this, add the info to the ticket and sync up with the assigned build person on the revision the needs to be committed. The assignee would update the ticket with the changes done (move it to resolved with relevant info and create clones if required).” (Wiki page PrH, P7).

Cloned MRs have been used in many projects: “We take a ticket from one platform branch, cloning them to another platform branch and linking them to the application feature. Basically it helps us to track back the issues.” (P2)

4.3.2 Cross-platform coordinators

Technical staff members who address cross-platform issues are critical to maintaining knowledge flow among projects. For example, a designated team consisting of developers from different Spark projects, such as SIP, H.323 and 1XC coordinate issues of common concern, such as changes in tools or process, buyback planning, and dependencies between projects. Other projects appointed a developer responsible for examining all components being developed for some other platforms. As an example, an appointed developer made the following observation: “...there is a Spark engine team who look on both sides, Windows and iOS to check for a bug across the platform . . .” (P8).

4.3.3 Cross-platform MR review meeting

The MR review meeting is an important standard practice in all Spark related projects as shown by this typical example. “Every week the scrum team should meet for about an hour to review the product backlog. This is the process of estimating the existing backlog using effort/points, refining the acceptance criteria for individual stories, and breaking larger stories into smaller stories.” (Wiki page – Project PrK).

Cross-project MR review meetings include project managers, technical managers and lead developers from each participating project. At each meeting, participants review requests from each participating team, review and create tickets, and make sure team members are aware of what is coming to them. “... every time there is an open code inspection everyone in the team can participate … we have a mandatory of minimum inspection from two reviewers when submitting our code …” (P10)

![Figure 3: Illustration of proposed measures](image-url)
Cross-project MR meetings help creating a culture of regular informal communication to clarify conflict issues, cloning issues and other issues as described in this example: “Synchronization is done by talking to the other team, such as the platform team. In the development level we are discussing everything, such as testing environment. For Tel Aviv site, every time they put a fix we need to test whether it introduces a bug to the application. Overall, there is a lot of coordination that happens on a daily basis. . .” (P4)

5. METRICS

Table 4 provides the definitions for the measures we used to quantify the concepts found in Phase 1 of the study and Figure 3 illustrates them.

### 5.1. Amount of coordination via cloning MRs.

To quantify the amount of cross-platform coordination via cloned MRs we use the number of cloned MRs (NclonedMR). As noted above, cloned MRs are an important bug-fix coordination vehicle with other projects. In the example in Figure 3,

\[
N\text{clonedMR(Project2)} : 0\text{(none for F2)} + 1\text{(MR2 for F3)} = 1
\]

The number of such MRs represents the amount of coordination and the fraction of MRs that is cloned for a project represents the intensity of coordination. We model the intensity of coordination in Section 6.2.

### 5.2. Extent of cross-platform development.

We used two measures to quantify the extent of cross-platform development: average number of forks per file (AvgForkFile) and fraction of files being changed in parallel (FracPara). In the example:
AvgForkFile(Project2) = \(2 \text{ (Project 1 and Project 3 for F2)} + 1 \text{ (Project 1 for F3)} / 2 = 1.5\)

FracPara is a ratio of the number of files in a project divided by the number of related files that were changed over the same period of time with at least one file from another fork. We use the ratio because projects with many files might also have many files being changed in parallel. AvgForkFile quantifies the extent of forking (that indicates potential need for parallel development) for the project code and FracPara quantifies the realized amount of parallel development.

5.3. Amount of redundant work.

We used two measures to quantify the amount of redundant work across platforms: the number of redundant commits (NRCom) and the number of redundant MRs (NRMr). NRCom is defined as a ratio of the redundant to total numbers of commits associated with all related files of a project. A redundant commit is a commit that is potentially repeated among related files. In the example,

\[
NRCom(\text{Project2}) = 3x(3-1)/3(F2)+2x(2-1)/2(F3) = 1
\]

For example, a bug-fix commit in one project may need and independent implementation in another project where developers may not be aware of the original fix. Such implementation would be clearly redundant because, if aware of the original fix, the second team could reuse it in their project if the code has not diverged too much.

NRMr is the number of redundant MRs modifying files related to files of a project. A redundant MR is defined as an MR that is potentially repeated among related files. In the example,

\[
NRMr(\text{Project2}) = (2 - 1)/2(F2) + (1 - 1)/1(F3) = 0.5
\]

Note, that redundant MRs do not represent MR clone relationship. Redundant MRs are defined as MRs changing a related file. A high number of redundant MRs implies a large number of tasks that are implemented independently in several projects.

Using these measures we pose the following hypotheses regarding factors that may affect coordination measured via MR cloning. In particular we expect more coordination measured via MR cloning with the following changes in multiplatform development:

- **H1** : Increase in the number of files changed in parallel

And with the following changes in the amount of redundant work:

- **H2** Increase in redundant commits.
- **H3** Increase in redundant MRs.

6. QUANTITATIVE FINDINGS

We first provide summaries of various measures and then test our hypotheses.
6.1 Descriptive analysis

The number of forks varies among Spark projects from 1 to 88. The median number of forks is 33. Figure 4 shows a distribution of forks per projects. 97% of projects in the Spark family have code related to other projects. 34 projects (79%) have code files spread over more than 10 forks. There are three projects with 70+ forks and all of them are common engine platforms that deliver code to many different applications across many platforms. There is a weak correlation (Spearman correlation coefficient = 0.286) between the number of files and the number of forks.

Table 5 shows that the fraction of parallel development in a Spark project ranges from 0% to 72%. There are only six projects where all related files were moved without a subsequent parallel change. The median value of code changed in parallel is 4.03%. There are two projects with more than 50% of their codebase being changed in parallel. They are relative young projects (less than a median project age) with less than 1000 files.

The number of redundant MRs and commits varies from 6.33 to 34708 (MRs) and 438 to 686792 (commits). This is not unexpected given the large variation of in numbers of forks per project.

6.2 How likely that an MR would be cloned?

In this section, we present the results of our logistic regression analysis modelling the intensity of coordination. The response variable is the likelihood that an MR will have a clone (MR clone-proneness). It implies the project’s propensity to coordinate work via MRs and reflects the fraction of work that is coordinated in this fashion.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Min</th>
<th>1/4Q</th>
<th>Median</th>
<th>3/4Q</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>NclonedMR</td>
<td>0</td>
<td>33.5</td>
<td>97</td>
<td>261.3</td>
<td>1017</td>
</tr>
<tr>
<td>AvgForkFile</td>
<td>0.02</td>
<td>0.49</td>
<td>1.06</td>
<td>2.73</td>
<td>63.41</td>
</tr>
<tr>
<td>FracPara</td>
<td>0</td>
<td>0.68</td>
<td>4.03</td>
<td>15.61</td>
<td>72.21</td>
</tr>
<tr>
<td>NRComm</td>
<td>438</td>
<td>3990</td>
<td>18243</td>
<td>91980</td>
<td>686792</td>
</tr>
<tr>
<td>NRMR</td>
<td>6.33</td>
<td>668.3</td>
<td>1941.8</td>
<td>5692.3</td>
<td>34708</td>
</tr>
<tr>
<td>Duration</td>
<td>428</td>
<td>834.5</td>
<td>1099</td>
<td>1395</td>
<td>1858</td>
</tr>
<tr>
<td>Nfile</td>
<td>81</td>
<td>613.75</td>
<td>2202.5</td>
<td>5180</td>
<td>132050</td>
</tr>
</tbody>
</table>
To obtain the response variable, for each MR affecting a project we assign 1 if it was cloned and zero otherwise. For the predictors described in Table 4, except for Duration, we take a log transformations (because all of them had highly skewed distributions). The 32008 MRs from 40 projects represent our observations. Due to high correlation among predictors NRComm, NredundantMR, we investigated them in separate models M1 and M2.

The logistic regression results are presented in Table 6. Residual deviance is 2605 and 2645 for M1 and M2, respectively. All independent variables are statistical significant at p < 0.01.

FracPara is the predictors that explain most variance in both models. Negative coefficient of FracPara shows negative relationships between with MR-clone proneness. The result differed from our initial expectation, hence, rejecting H1. This implies that the extent of parallel development decreases the probability of having a cloned MR. Perhaps, forking and more parallel development between independent projects leads to less coordination needs via cloned MRs.

Positive coefficients of NredundantCommit and NredundantMR show a positive relationship between these variables and MR-clone proneness, confirming our hypotheses H2 and H3. This observation implies that the amount of redundant work increase the probability of having a cloned MR.

The model indicates that the project duration (reflecting project’s age) is one of the most important predictors of probability leading to a cloned MR. Interestingly, the model shows a negative relationship between project duration and MR clone-proneness. This may reflect the increasing propensity over time to use cloned MRs as a coordination mechanism as existing mechanisms of coordination are no longer sufficient.

### 7. RELATED WORK

#### 7.1. Forking and coordination in multi-platform development

Joorabchi et al. explored challenges in developing mobile apps and revealed several issues about multiple platform development, such as platform technical variety, maintenance of common core codebases, different scripting language among platforms [11], growing testing space [1, 13], and cross-platform security flaws [1]. In this study we confirmed that challenges occur between tester-developer and manager-manager.
Table 7: Hypothesis testing

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Hypotheses</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross platform dev.</td>
<td>H1</td>
<td>Reverse direction</td>
</tr>
<tr>
<td>Redundant work</td>
<td>H2</td>
<td>Expected direction</td>
</tr>
<tr>
<td>Redundant work</td>
<td>H3</td>
<td>Expected direction</td>
</tr>
</tbody>
</table>

In 2001, Perry et al. conducted an observational case study of parallel development in a legacy multiplatform software system and found that current tools, processes and project management support for this level of parallelism is inadequate [19]. Our study shows that with support of collaboration tools and modern VCS, development and coordination across multiple platforms is still a challenge.

Previous studies suggested inner-project branching strategies to deal with technical issues, such as new branch for new release, service patch and testing [7]. In this study, we found that inter-project forking is driven by organization strategy and policy. Previous studies found that the main reasons open source projects fork are technical variation, discontinuation of original project, community-driven, sustainability and legal issues [18, 20]. Similar to open source projects, forks are used to divide work into different organizational teams and to deal with technical variations. Unlike open source forks, commercial forks are driven by different schedule release, balance between coordination and development effort and general product development strategies.

Kotlarsky et al. defined an integrative framework with four type of coordination mechanisms: coordination by organization design, work-based coordination, representations of work-in-progress, inter-personal coordination and technology based coordination [14]. Our study reveals three types of mechanisms: consistent usage of collaboration tools across platforms (Cloning MR in Jira), organization design (cross platform coordinators) and work-based coordination (review meetings). While cloning MR is the main approach to identify and trace duplicated bugs, it is conducted in a rather ad hoc manner. Organization design and work based mechanisms play a role in shaping process and practices in adopting technology-based coordination mechanisms.

7.2 Coordination needs in parallel development

Similar to findings in open source projects [20], most forks evolve in parallel to the original projects. In our study, 65% of forks are active, compared to 73.4% of continued forks in open source projects [20]. In our case, there is a large fraction of MR cloning in Spark (83% of its peak). Swcharz et al. investigated code clone across repositories in an open source ecosystem and found that 14 % of all methods are copied from another package in another project [25]. While we did not directly measure the amount of code clone across forks, the median value of fraction of cloned MRs is 29.16%. Perry et al. showed that a legacy system might have 3-4 releases undergoing development at any given time [20]. Our studies reveals some period of time Spark projects has all 43 forks changed.

Bird et al. proposed a relationship between goals and virtual teams on different branches [5]. The authors suggested that if two branches have similar goals, they would also have similar virtual teams or be at risk for communication and coordination breakdowns with the accompanying negative effects. Shihab et al. found that branching does have an effect on software quality and that misalignment of branching structure and organizational structure is
associated with higher post-release failure rates [26]. Cataldo et al. measured Coordination congruence and found the mismatch in logical coordination requirements and actual coordination via MRs leads to lower software quality. In this study, we found that amount of actual coordination via cloning MR is reduced by deferring coordination requirements using independent forks. However, the actual coordination is increased in forks with high amount of redundant work.

7.3 Threats to validity

While designing and conducting this study, various conscious decisions were taken to strengthen the validity of results. We used a suggested checklist [21] to ensure that we had addressed all the critical requirements of case study design including aims of the study, defining the case, unit of analysis and data collection methods among others.

Construct validity refers to the relations between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. In this study, perspectives of multiple platform development is quantified based on grounded qualitative findings, which reduce this validity threat.

Internal validity refers to the existence of causal relationships between treatment and outcomes. One way to reduce this threat is to include potential confounding factors that may affect the relationship into the model. We include project duration and several other predictors that may affect the investigated relationship into the model.

External validity concerns about the generalization of observed results to a larger population. The case in our study could represents a large and complex product family in software and telecommunication industry.

Credibility concerns the confidence in the result. We adopted several threat mitigation strategies, such as sending interview transcripts to interviewee for correction; involving three researchers on study design and review, methodological triangulation and detailed documented steps of data collection, processing and analysis. Thus we believe that the resulting detailed protocol serves as a good means to replicate this study.

8. CONCLUSIONS

Our study provides a better, more comprehensive understanding on the state-of-the-art of multiple platform development. The Spark family has a large amount of parallel development and potentially a lot of redundant work. While intra-project branching is driven by technology, we found that inter-project forking is driven by market and management strategies. Our study also reveals cloning MRs is an important coordination mechanism in the Spark family. Logistic regression analysis suggested that coordination needs decrease in parallel development among independent projects but increase in projects with a large amount of redundant work.

We reveal some opportunities for future research in this area. In this study, we only investigated the extent of forking activities and redundant work via commits and MRs. In future work, we would like to address code divergence and assess the number of identical files between them in a more detail manner. In particular, we would like to investigate if there is any difference between a file that diverges much compared to a file that does not in term of
software quality and coordination efforts. Last but not least, this study revealed two important coordination mechanisms for multiple platform development. However, the effectiveness of these mechanisms in addressing coordination needs is beyond the scope of this study. Future work can focus on evaluating the effectiveness of these mechanisms.

We also derive some implications that practitioners can consider when taking action in multiple development platforms. The primary value of VCS is to allow parallel development of branches to delay the merge time. In order to reduce technical debt at system level, periodical synchronization of forks is necessary. Testers need to understand that there might be a lot of redundant functionalities among different platforms. A look beyond scope of a single platform may help to reduce the amount of duplicated test cases. Product line managers need to enforce formal processes for cloning MRs across codebases, such as periodical review of cross platform issues. Last but not least, it is helpful to automatically detect code divergence and propagate changes in related files to relevant developers.

REFERENCES


[27] M. Swider. Microsoft highlights 299m skype users, 1.5b halo games played, 2013.


Paper MP8 - A longitudinal case study of coordination in multiple platform development

Longitudinal study on coordination and development in multiple platform development

Anh Nguyen Duc\textsuperscript{1,2}, Audris Mockus\textsuperscript{2}, Randy Hackbarth\textsuperscript{2}, John Palframan\textsuperscript{2}

\textsuperscript{1} Norwegian University of Science and Technology, Sem Saelands vei 7-9, Trondheim, Norway
\textsuperscript{2} Avaya Labs Research, 211 Mt Airy Rd, Basking Ridge, NJ, USA
anhn@idi.ntnu.no,\{audris,randyh,palframan\}@avaya.com

ABSTRACT

[Motivation] With the proliferation of desktop and mobile operating system and hardware platforms the development and maintenance of identical or similar applications on multiple platforms is urgently needed. [Objective] We aim to understand code sharing and coordination in software system developed for multiple platforms. [Context] We study enterprise software product in the voice, video, and instant messaging domain deployed to more than 25 software/hardware platforms over 10+ years. [Method] A longitudinal case study using semi-structured interviews, project wikis, VCSs and issue tracking systems was conducted to understand development challenges, reasons for codebase divergence, coordination mechanisms and to quantify these aspects. [Results] We found diverged codebases to be difficult or impossible to avoid because of release schedule, feature variation, coordination overload, and organizational structure. A typical codebase had 33 related forks and 61\% of related code files under parallel development. Developers of multi-platform systems have to cope with coordination debt, duplicate work, and coordination across global boundaries. [Conclusions] Critical tasks involve merge and coordination of diverged codebases and, among other things, require focus of cross-platform coordination effort on the files under parallel development. Our findings suggest that more research is needed to support this rapidly growing field of software development.

1. Introduction

Modern society depends heavily on complex heterogeneous computing systems in a variety of hardware infrastructures, ranging from personal computers, workstations, and cloud servers to telecommunication and mobile devices running on a variety of operating systems and interface environments. There is a common and increasing demand for similar functionalities in different configurations of software and hardware platforms. Typical examples are web browsers, instant messaging clients, communications, and other mobile applications.

Software application providers that want to reach a large user audience must develop their apps to run on multiple platforms. Multi-platform development, however, has several implications for the application developers that make multi-platform development and support difficult.

First, the maintenance and upgrades, which account for two-thirds of a typical software project’s lifetime cost (Bishop and Horspool 2006), might take even more effort in a cross-platform system development. This is because defect fixes must be propagated across
multiple versions of a software application. For example, a recent urgent Apple defect, which was fixed for iOS, but remained unpatched for Mac OS (Moren, Caldwell, et al. 2014), illustrates the difficulty of maintaining consistent code quality across platforms.

Second, not many developers are intimately familiar with multiple platforms, and as a result, it is often necessary to have different individuals and teams implement the same product for each platform. The teams implementing on different platforms need to be able to effectively coordinate their work.

Third, multi-platform applications need to maintain inter-operability, provide consistent performance and a consistent look-and-feel of the application across different platforms. For instance, a GUI has to be implemented in platforms-specific frameworks, but the application features must work in a similar fashion across all platforms. Therefore, development teams have to conform to different requirements and constraints for each platform (e.g., iOS, Android, Windows 7, etc.), for variations in hardware (e.g., HTC, Google, Samsung), and, potentially, for different communication protocols (e.g. AIM, XMPP, SIP, H323).

It appears that traditional software engineering approaches may not be directly applicable in a context with many challenges that are unique to multi-platform development. The current literature on multi-platform development is highly pragmatic, with guidelines and technical solutions (Arisholm, Briand et al. 2004, Bird, Zimmerman, et al. 2011, Bishop and Horspool 2006, Bufenbarger et al. 1999). One major focus is to design a product family architecture that goes across multiple platforms (Jiao et al. 2007) and has strategic planning of a product family that aligns with economic and market demands (Halman et al. 2003). To the best of our knowledge, however, no empirical study has been published of software development practices and coordination needs supporting multi-platform development.

To understand how software intensive organizations resolve development and coordination problems of multi-platform development in practice, we performed a mixed-method longitudinal study of an instant messaging system at Avaya. In the qualitative part of the study, we explored the state-of-the- development challenges and coordination mechanisms when working with the system. In the quantitative part we constructed a network of related file versions across 43 instances of version control systems (VCS) used by 25 different software and hardware platforms. We used the network to quantify the extent of parallel development and coordination activities. The adoption of coordination mechanisms are assessed by association with customer related defects. Our contributions include the following.

- We describe what happened in multi platform development, the state-of-the-art of challenges; practices and reason for codebase forks.
- We propose measures of the extents of parallel development, the amount of coordination effort, and the extent of redundant work at the project level.
- We assess the relationship between proposed measures and customer driven software quality.
- We provide recommendations for managing and coordinating the development of multiple platform systems.
2. Background and terminology

We start by defining the terminology used in this paper.

2.1. Sociotechnical dependencies and coordination

Key phenomenon in modern software development is coordination of development tasks across global boundaries (Herbsleb and Moitra 2001; Holmström, Conchúir, et al. 2006; Noll, Beecham, et al. 2010; Prikladnicki, Audy, et al. 2003; Nguyen-Duc, Cruzes, et al. 2014). Coordination is understood as an action of orchestrating each task and organizational unit, managing inter-dependencies among them, so that they all contribute to the overall objective. If tasks carried out at different globally distributed locations shared no dependencies, asoftware project would not pose significant challenges as engineers would not need to communicate with or even know of the existence of other locations.

No single widely accepted definition of coordination exists. Coordination is a multidisciplinary phenomenon, which is investigated from system design and modularity perspective (Conway (1968); Parnas (1974); Sosa &Eppinger (2004); Baldwin & Clark 2006), organizational structure (Malone and Smith, 1987; March and Simon, 1958; Mintzberg, 1980; Thompson, 1967), and knowledge management (Kotlarsky, Van Fenema, et al. 2008)

We adopted a multidisciplinary theory of coordination by Malone and Crowston (Malone and Crowston 1994). Coordination is defined as: “Activities required to maintain consistency within a work product or to manage dependencies within the workflow”. The main idea in Coordination Theory is that coordination is needed to address dependencies, which are the constraints on action in a situation. If there are no dependencies, there is nothing to coordinate. But if a task depends on the result of another task, or if the execution of a task depends on the expertise of the person that performs the task, the dependencies must be managed.

Dependencies in software development can be traced back to the technical dependencies among software components. Traditional software measures have captured logical and syntactic dependencies by measures of e.g., cyclomatic complexity, cohesion and coupling (McCabe 1976; Chidamber and Kemerer 1994). Other types of technical dependencies include data-related dependencies (i.e. a data structure modified by a function and used in another function) (Arisholm, Briand et al. 2004) or functional dependencies (e.g. method A call method B) (Joorabchi, Mesbah, et al. 2013) and logical dependencies (e.g. files being changed together) (Cataldo, Mockus et al. 2009).

Coordination mechanisms are approaches to manage dependencies among software artifacts and project stakeholders, and is an important research topic (Kraut and Streeter 1995). Thompson classified coordination activities to be either of an organic or mechanism type (Thompson 1967). Organic coordination is the use of lateral communication means to coordinate activities, while mechanistic coordination is the use of vertical communication means to coordinate activities in a programmed way, such as task organization, role assignment, schedules, plans, division of labor, project controls and specifications, routine meetings and status checks. The resolution of coordination needs might occur in time or delayed by modularizing task, allowing isolated development and deferred code integration.
Coordination debt results from a delay in coordination to save time in the short term but results in more effort in the longer term. Coordination debt is a variation of technical debt, a metaphor introduced by Ward Cunningham in his experience report (Cunningham 1993). The metaphor describes a phenomenon in which technical quality issues in a software system can lead to future problems if not resolved immediately. To date, the technical debt concept is gaining traction as a way to focus on the tradeoff between the short-term payoffs (such as a timely software release) of delaying some technical development activities and the long-term consequences of those delays (Brown, Cai et al. 2010). The metaphor highlights that, like financial debt, technical debt incurs interest payments in the form of increased future costs owing to earlier quick and dirty design and implementation choices.

The type of debt can be helpful to tailor debt payment to critical quality characteristics of interest. For example, known defect debt (known latent defects that have not been fixed) may be differently perceived in quality critical software applications (Guo, Seaman et al. 2011). Other known TD types are: design debt (an imperfection of the software’s design or architecture negatively affecting future maintenance) (Brown, Nord et al. 2010), documentation debt (missing, outdated, or incomplete documentation), and testing debt (missing test cases, test cases that are not executed, or missing test plans). In this study we explored a new type of technical debt, namely coordination debt (discussed in Section 4.3).

2.2. Boundary spanners

As software project involves multiple internal and external roles, such as project manager, software developer, testers, customers and 3rd parties, coordination of development activities often require a communication across these individual groups. IS literature has emphasized the importance of relying on individuals to perform a boundary spanning role (Davenport and Prusak 1998, p. 68; Hargadon and Sulton 1997; Pawlowski and Robey 2004). According to these works, a boundary spanner is viewed as a role linking “organizational structure to environmental elements, whether by buffering, moderating, or influencing the environments”. Boundary spanners also translate and frame information from one environment to another in an effort to promote coordination (Tushman and Scanlan 1981, Aldrich and Herker 1977, Brown and Duguid 1998). Boundary spanners can be formally assigned to take charge of coordinating other people, or informally emerged by supporting other team members. In this study, we operationalize cross platforms spanners to be developers who contributed code to several multiple cross platform codebases. These developers are supposed to have technical knowledge about these platforms, and hence play a role of technical boundary spanner.

In Software Engineering, the role of boundary spanner in coordinating team activities is not much explored. Boundary spanner can be taken by a requirement analyst, a project manager or a special engineer to cross boundaries between customers and development teams. (Sowe, Angelis, et al. 2001; Marczak, Damian, et al. 2008; Boden, Avram et al. 2009). Depending on the organizational role it takes, a boundary spanner might have different abilities and perform different kind of coordination activities in different context. Pawlowski et al. described an ability to facilitate sharing of expertise, integrating communication of the team (Pawlowski and Robey 2004). Faraj et al. highlighted an ability of boundary spanners to protect inner team and obtain political support (Faraj and Yan 2009). Strode et al. looked at boundary spanning roles as a coordination strategy in four co-located projects (Strode, Huff et al. 2012). Krishnan et al. examined the usefulness of boundary spanning in collaborating distributed teams via project social capital perspective (Krishnan and Ranganathan 2009). In our study,
we explore the role of cross platform spanner in a specific context of detecting duplicated works and resolve them.

2.3. Configuration management in multiple platform system

A configuration management system is a central infrastructure of modern software development. We summarize several important concepts that relate to configuration management:

• **Version Control System (VCS)** manages changes to computer code. Each temporal instance of a file is uniquely identified and retrievable. Examples are IBM Rational ClearCase, Apache™ Subversion®, Concurrent Versions Systems (CVS), and GIT. In this study, we collected data from two VCS, namely Subversion and GIT.

• **Project** has a its own code repositories (codebases) with active code commits and modification requests and results in one or more releases of a product. A project can have more than one codebase due to, i.e. upgrade of VCS, code transfer and movement.

• **Fork** is a clone of an entire, or large logical piece, of a product’s code repositories that has subsequent and independent development from the original project that results in a distinct project. Fork can be created easily via Git. In this study, we focus on forks, which are created to delivery new functionality or defect fixes to different platforms. We did not consider numerous branches created for the same platform as forks.

• **Commit** is a set of changes to one or more file tracked in a Version Control System. Commits to Version Control Systems (VCSs) are associated with a MR identifier in the Avaya projects that are the subject of this study.

• **Issue tracking system**, such as JIRA maintains a list of uniquely identified issues associated with a project that include descriptions and measures states, and resolution of the issues. An issue tracking system is used by project teams to capture and organize issues, to assign work and to monitor team activity. In this study, we investigated the usage of a issue cloning feature in JIRA as a coordination mechanism.

• **Modification request** (MR) is a piece of work tracked by an issue tracking system. In our study we focus on issues that are defects, which can come from a development team, a product team, a system verification team or customers. Customer related defects are defects that are found in customer environments. We use customer related defects in a later phase of this study to validate effectiveness of coordination mechanisms. We also use the term JIRA ticket to refer to MR.

• **Multi-platform software** refers to the ability of software to operate on more than one platform with identical (or similar) functionality (Bishop et al. 2006). Platform can refer to (1) different types of operating systems (e.g., Windows, FreeBSD, Linux, Mac OS X, and Solaris systems), (2) types of processors (e.g., x86, PowerPC, SPARC or Alpha), (3) communication protocol (e.g. SIP, RTM, AIM and H323) or (4) types of hardware systems (e.g., iphone, ipad, android phone and windows phone). A main issue of multi-platform software development is that the software/ components often need to be developed in parallel. These issues will be explored in Section 4 and Section 5 of this paper. We present the definition of concepts about multi-platform software development:

• **Directly related files** are files that had identical content in the past (Mockus et al.
Figure 1 describes a relationship between file F1, F2 and F3. Dashed line represents a cloning relationship (identical copy), i.e. F2a clones F1a. Grey line represents an evolution of files by their commits. In the example, F1 and F2 and F1 and F3 are directly related because they were cloned from each other at some point in their version history. We refer to the transitive closure of directly related files as related files. In particular F1 is related to F2, and F3, and F2 is related to F3 but F2 is not directly related to F3.

- **Files under parallel change** are files that are changed together during some interval of time and are related. In Figure 1, File F1 and F2 are under parallel change in the beginning of F1 history and later file F1 and F3 are under parallel change.

- **Cloned MR** is a replica of the original issue, containing the same information stored in the original issue — e.g. Summary, Affects Versions, Components, etc. The cloned MR can also be linked to the original MR. ClonedMRs are used as a coordination mechanism to track duplicated development efforts among different projects.

- **Redundant commit**: for N related files, a commit to one of the files is a redundant commit to the commits to the N-2 other files. The number of redundant commits is defined and calculated as in Section 4.4.1.

- **Redundant MR**: each commit to file has associated MR. For N related files, the MR for one file is a redundant to MRs for the N-1 remaining files. The number of redundant MRs is described in detail in Section 4.4.2.

3. RESEARCH METHOD

3.1. Research design

A current research focus in Avaya R&D is to improve software risk management practices within R&D organization. In numerous projects in Avaya, it was observed that only 1% of project files are involved in more than 60% of the customer reported defects (CFDs) (Hackbarth et al. 2013). Thus focusing quality improvement on such files can greatly reduce the risk of poor product quality. As part of this research, we aimed at exploring the current development and coordination practices in inter-dependent projects. The overall goal of this research is (1) to give the Avaya R&D organization an overall picture of development and coordination in one of their set of multiple-platform codebases, (2) to introduce solutions to improve the state of software in Avaya and (3) to know it.

![Figure 1: Illustration for related files](image)
3.1.1. Action research

The research methodology employed in this study is action research with a five-phase cyclical process. Action research is most widely adopted in the social sciences (Greenwood and Levin 1998, Davison, Martinsons et al, 2004). The main objective is to solve a real-world problem while simultaneously studying the experience of solving the problem. The problem in our case is the challenge of maintenance and coordination of related files across multi-platform projects. The embedded research team in the R&D organization allows access to data materials and the ability to introduce changes to the organization. Action research merges research and practice, as the research team is expected to develop recommendation and solutions that help solve some of the challenges with multi-platform codebases.

The method consists of five steps as shown in Figure 2 (Susman and Evered 1978):

- **Diagnosing** refers to the joint (researcher and practitioner) identification of actual problems and their underlying causes. During this phase, researchers and practitioners jointly formulate a working hypothesis of the research phenomenon to be used in the subsequent phases of the action research cycle. In this research project, we performed a longitudinal case study to identify ongoing challenges with multiple platform development.

- **Action planning** is the process of specifying the actions to improve the problem situation. Findings from different phases of the case study are reported as an action planning, including recommendation for practice changes and adoption of tools.

- **Action taking** refers to the implementation of the intervention specified in the action-planning phase. In this research project, we applied the prototype in some real projects and collected feedback from project members.

- **Evaluating** entails the joint assessment of the intervention by practitioners and researchers. **Specifying learning** denotes the on-going process of documenting and summarizing the learning outcomes of the action research cycle. These steps are not included in this phase of the research project.
We carried out the research in an iterative manner. The research team proceeds with a practical focus as it keeps working and diagnosing new issues and practices with multi-platform development, introduces changes and quantifying the impact on actual development. The advantage of this continuous cycle is to allow fast delivery of practitioner driven research outcomes.

3.1.2. Case study and mixed methods

We performed a longitudinal study for continuous diagnosis of multi-platform development in Avaya through several iterations. Case study research is concerned with the researcher gaining an in-depth understanding of particular phenomena in real-world settings (Yin 2002). According to Yin, case studies are the preferred research strategy “...when a «how» or «why» question is being asked about a contemporary set of events over which the investigator has little or no control.” The case represents typical multi-platform software development with involvement of teams across geographical and organizational boundaries. The case study is longitudinal, including different phases of discovering development practices in multi-platform systems. At this phase we focused on coordination practices and approaches to manage diverged codebases. The result of this phase leads to some recommendations for changes in development practices.

We adopted mixed methods to collect data from different sources and to describe the comprehensive picture of coordination and maintenance in multi-platform development. Mixed methods research involves combining techniques, methods, approaches and language of both quantitative and qualitative traditions in a single study (Johnson and Onwuegbuzie et al. 2004). The reasons to combine quantitative and qualitative approaches to research include the following:

- triangulation, by investigating the issue from different perspective and then converging the results (Cresswell 1999);
- complementarity data: qualitative data describes the phenomenon, and quantitative data add numbers in words or pictures (Greene et al. 1989, Johnson and Onwuegbuzie et al. 2004);
- the results of one method may lead to a further step in the research with the use of another method, or create research questions and hypotheses for another method study (Greene et al. 1989);
- mixed methods study allows reaching a larger audience (Dörney 2007);
- the research claims become stronger if the data come from a variety of methods, and ensures “less waste of potentially useful information” (Gorard et al. 2004);
- mixed methods research allows investigation of a wider range of research problems, i.e. both exploratory and confirmatory questions (Schulenberg 2007);

3.2 Research setting

Avaya is a large provider of call-center and business communication software that are in embedded and standalone products. Avaya has numerous software projects with approximately 3000 developers spread across 13 time zones in North America, South
America, Europe and Asia. R&D employs many virtual collaboration tools such as JIRA, Confluence, Git, WIKIs, Sharepoints, virtual document reviews, and Crucible.

A typical development team in Avaya uses Scrum-like development methodologies with a 4-week sprint. Sprint MRs are assigned to a developer/tester who can complete it in a specific timeframe. If all MRs are finished before the end of the sprint, other MRs will be assigned from a backlog. Code delivery will always be accompanied by an MR ticket. The ticket may be a result of a defect or a new feature. Several coordination approaches have been used, such as a (1) daily scrum meeting, (2) sprint planning meeting, (3) sprint review, (4) backlog review meeting, and (5) scrum of scrums meeting.

Product quality is owned by each Avaya development team and is verified by a separate system verification (SV) team. Some development teams include specialized sub teams that are focused on development testing. These sub teams include system integration teams (SIT) that closely collaborate with the development team, running tests such as unit tests and new feature tests and a product verification (PV) team that is responsible for quality assurance through the integration of modules across a product.

The SV team is responsible for verifying user scenarios supported by the product, capacity and performance testing, and regression testing among other responsibilities. The SV team certifies when the product is ready for alpha and beta deployment. Another organizational unit, the SIL System Integration Lab (SIL), is responsible for cross product testing across Avaya products.

Avaya development teams use a version control system, an issue tracking system (typically JIRA) and a Confluence wiki among other collaboration tools. Confluence provides an overview of the site, gives access to the spaces that you have permission to view, and displays a few different lists of the most recently updated content. JIRA provides issue tracking and project tracking for software development teams to improve code quality and the speed of development. JIRA provides a mechanism to clone or copy an issue, creating a duplicate of an issue within the same project. Besides, JIRA has a mechanism to link an issue to existing issues, for instance: (1) an issue that relates to another, (2) an issue that duplicates another, (3) an issue that may block another. In Avaya R&D, clone plus plugin is updated regularly to ensure proper functions of cloning mechanism via JIRA.

Our study focused on a 10+ year old software component, the so-called Spark engine. As a software platform, Spark provides a consistent set of signaling platform functionalities to a variety of Avaya telephone product applications, including those of third parties. The current evolution of Spark is a client platform that provides a signaling manager, a session manager, a media manager, an audio manager and a video manager. The Spark engine started in 2005 and has forked into many different codebases to support different combinations of hardware platforms, software (iOS, Android, and Windows) and communication protocol (SIP and H323).

3.3 Data collection

A series of meetings was conducted with different members of R&D to negotiate access to the organization and available data repositories. Several discussions and interviews with internal stakeholders were conducted to identify relevant projects for analyzing multi-platform systems. The research was conducted between December 2012 and March 2014.
Our initial focus was to understand the context of multi-platform development, including technical and management challenges, and current development and coordination practices. A range of data collection methods were used to achieve familiarity with the organizational context and participants, including conversations and interviews, and review of relevant documentation and project artifacts.

Our second focus was quantifying important concepts found during the previous phase. Data was collected from different sources. File information, such as source code, file history and its commits was collected from version control systems (e.g., SVN and Git). Modification requests (MR) were extracted from JIRA.

**Project selection**: The selection of relevant projects (which used the Spark engine and involve cross-platform development) is difficult as project boundaries are often not visible at the code repository or issue tracking system level. Besides, many projects have little written information related to multiple data repositories. We found that many projects have a long history of merging and transferring source code. We utilized expert opinions from those who worked on Spark-related projects for a long time to identify the relevant projects. The projects chosen for interviews are shown in Table 1.

**3.3.1. Semi-structured interview**

Semi-structured interviews are the primary data source in this phase. The interview guide was designed using the checklist from the literature review and revised multiple times by the authors. The interview guide includes a list of planned questions and interview topics aimed at ensuring their systematic coverage across interviews. However, the interview is flexibly conducted to allow for improvisation and exploration of emerging issues (Runeson and Host 2009). The interview guide consists of questions about (1) challenges of working on inter-platform teams, (2) state-of-the-art of codebase divergence, (3) coordination mechanisms, and (4) codebase merging activities. The detail guideline is given in Appendix A.

**Interview selection**: Interview subjects were selected based on quantitative analysis. As we would like to understand the development challenges and coordination practices in a multi-platform project, interviewees should have experience with this area. The key member of a Spark project is first identified by reference from our researcher, who has extensive knowledge about Spark projects. We obtained multiple viewpoints of the study topic by having interviewees from different project positions, such as program managers, team leaders, developers, and testers. We looked at a list of team members from Project wiki page to get

---

**Table 1: Projects included in interview**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrA</td>
<td>Engine components in a conference phone C1 using protocol P1</td>
</tr>
<tr>
<td>PrB</td>
<td>Engine components in a conference phone C2 using protocol P1</td>
</tr>
<tr>
<td>PrC</td>
<td>Engine components in a conference phone C3 in iOS</td>
</tr>
<tr>
<td>PrD</td>
<td>Components for phone C4 apps in mobile devices</td>
</tr>
<tr>
<td>PrE</td>
<td>Engine components in a conference phone C3 in Windows</td>
</tr>
<tr>
<td>PrF</td>
<td>Engine components in a conference phone C3 in Android</td>
</tr>
<tr>
<td>PrG</td>
<td>Engine components for phone apps using protocol P2 in iOS</td>
</tr>
<tr>
<td>PrH</td>
<td>Core engine functionalities for a set of conference phones</td>
</tr>
<tr>
<td>PrI</td>
<td>Engine components in a conference phone C3 in iPad</td>
</tr>
<tr>
<td>PrJ</td>
<td>A set of components for conference phone C5</td>
</tr>
</tbody>
</table>
contact information. One problem is that many of these lists were out of date and the team member no longer worked in the projects or even in Avaya. To overcome this issue, we used a tool developed by one of our researchers that visualizes active developer list with their information, such as locations, number of commits and projects they worked for. Using this, we identified the main contributor for each project. Moreover, we also adopted a snowball approach by asking each interviewee for references at the end of each section. The list of interviewees is given in the Table 2. We used the saturation approach for interview selection. We continued searching for new interviewee until no new information was added.

The interviews were typically audio-taped (with permission), transcribed and reviewed by interview participants. Table 2 describes the list of interviewees in this study. We conducted 10 semi-structured interviews with different project roles, such as development manager, development lead, developer and researcher. Each interview lasted from 30 to 60 minutes, depending on time availability of interviewee.

3.3.2. Data repository
The primary data sources in this phase were version control systems (Git and SVN) and the issue tracking system (JIRA). The data available in Spark projects was huge due to the long history and size of the group of projects. We started from data integrated over all Avaya projects as described in previous studies (Mockus et al. 2007, 2009, 2013, 2014). Data extraction consists of the following steps:

- Retrieve the raw data from the projects’ version control systems (e.g., SVN, Git) and issue tracking systems (e.g., JIRA), or search relevant information from the web interfaces of these systems. For example, SVN changes can be obtained via the `svn log` command, and commit history of files is stored in SVN and Git log files.

- Clean and process raw data to remove artifacts introduced by underlying systems. Verify completeness and validity of extracted attributes by cross-matching information obtained from separate systems.

- Construct meaningful measures that can be used to assess and model various aspects of software projects. In particular, we quantified the extent of forking activities, amount of coordination via cloned MRs and the extent of potentially redundant development.

<table>
<thead>
<tr>
<th>Table 2: Interview list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
</tr>
<tr>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P4</td>
</tr>
<tr>
<td>P5</td>
</tr>
<tr>
<td>P6</td>
</tr>
<tr>
<td>P7</td>
</tr>
<tr>
<td>P8</td>
</tr>
<tr>
<td>P9</td>
</tr>
<tr>
<td>P10</td>
</tr>
</tbody>
</table>
Table 3: Quantitative data

<table>
<thead>
<tr>
<th>Item</th>
<th>Min</th>
<th>1st Quart.</th>
<th>Median</th>
<th>3rd Quart.</th>
<th>Max</th>
<th>Skewness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project duration (days)</td>
<td>375</td>
<td>725.5</td>
<td>1066</td>
<td>1309.5</td>
<td>1770</td>
<td>0.11</td>
</tr>
<tr>
<td>Number of files (files)</td>
<td>83</td>
<td>579.5</td>
<td>1208</td>
<td>4369.5</td>
<td>46958</td>
<td>2.92</td>
</tr>
<tr>
<td>Number of forks</td>
<td>1</td>
<td>19</td>
<td>33</td>
<td>51.5</td>
<td>215</td>
<td>2.40</td>
</tr>
<tr>
<td>Number of commits</td>
<td>1217</td>
<td>7035</td>
<td>28392</td>
<td>91101.5</td>
<td>716148</td>
<td>2.53</td>
</tr>
<tr>
<td>Number of code MR</td>
<td>2</td>
<td>76</td>
<td>205</td>
<td>575</td>
<td>4599</td>
<td>2.46</td>
</tr>
</tbody>
</table>

The data unifies information from most of version control and issue tracking systems in Avaya, but we still need to select the projects related to the client’s communication. Three authors of the papers had insights on the products, which helped to identify 67 relevant codebases. We interviewed key stakeholders to confirm and complement the codebase list. To select and link relevant data, we wrote shell, Perl, and R programs to extract and calculate the necessary measures.

After all data was extracted, we performed data cleaning to remove inactive repositories (repositories without changes in the last 12 months) and moved repositories (repositories moved from one VCS to another VCS without subsequent development activities in the original VCS). 20 inactive repositories and 4 moved repositories were removed. Table 3 provides a summary of the final dataset of 43 project repositories for the 25 platforms. The median duration of a Spark project is 1066 days and typical size is 1208 files. A typical Spark project has 205 MRs and 28392 code commits. For instance, the median number of forks from a Spark projects are 33 with the maximum value is 215. The dataset is given in Appendix C.

3.3.3. Documentation

Another data source used to supplement interviews were project documents, such as requirement descriptions, product and process descriptions, project wiki pages, mailing lists and summaries of JIRA tickets. All of these documents were made available in Confluence. Project policy, planning and process documents were reviewed so that the field researcher could become familiar with the development processes, practices and terminology. Confluence gives an overview of projects and features to be delivered. A lot of information about team structure and role assignment can be found in the project wiki page. For example, the project tracking system and task allocation plan gives an overview of project progress and identifies current challenges. The mailing list and JIRA information on the wiki page provides communication data among developers.

3.4. Data analysis

3.4.1. Thematic analysis

We used thematic analysis, a popular method to analyze qualitative data, to identify concepts and their relationship from interview data. We analyzed the data in three steps. First, we read all interview transcripts and relevant documents, and coded them according to open coding (Miles and Huberman 1994). Each segment of text that expresses either challenges or practices in multiple development platforms was labeled with an appropriate code by open coding. After that, we grouped the relevant codes as a higher-order code using axial coding (Miles and Huberman 1994). Finally, we performed selective coding in secondary data materials, such as project wiki and MR descriptions.
3.4.2. Regression model

Based on findings in qualitative phase, we proposed several measures of development and coordination in multiple platform system. We also investigated the relationship between these measures and software quality with a regression model. Regression models are generally used in empirical studies to model an outcome of a response variable (e.g. model the number of post-release failures) or to model the relationship between an observed phenomena (represented by the model independent variables) and an observed outcome (represented as the dependent variable). We used multivariate regression models to study the relationships between the amount of coordination, redundant work and the extent of parallel development. We did not plan to predict any specific project outcomes but we were interested in evaluating the effectiveness of coordination mechanisms and how they related to the quality of the product. In particular, we wanted to examine the relationship among coordination effort, extent of parallel development and redundant work.

3.5. Threat to validity

While designing and conducting this study, various conscious decisions were taken to strengthen the validity of results. We used a suggested checklist (Runeson and Höst 2009) to ensure that we had addressed all the critical requirements of case study design including aims of the study, defining the case, unit of analysis and data collection methods among others. We considered issues from using both qualitative and quantitative data.

**Construct validity** refers to the relations between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. The independent variables in our work operationalize three concepts, the extent of parallel development, coordination effort and redundant work. The measure of these concepts is based on grounded qualitative findings as well as literature, which reduce this validity threats. The dependent variables in our study are measures of software quality, i.e., likelihood of having a customer related defect. The measure is defined as the probability of finding a defect by customers. Assuming that these defects were properly found in field test, the dependent variables are also satisfactory.

**Internal validity** refers to the existence of causal relationships between treatment and outcomes. One way to reduce this threat is to include potential confounding factors that may affect the relationship in the model. We include project duration and several other predictors that may affect the investigated relationship into the model.

**External validity** concerns the generalization of observed results to a larger population. The case in our study represents a large and complex product family in the software and telecommunication industry. Since we have performed our empirical analysis on one data set, the external validity of our work is threatened. Specifically, we cannot generalize the results of using the specific set of predictors, which we selected to other systems. However, we can be confident that the model results are statistically significant; therefore, the modeling approach would be applicable when dealing with subsequent versions of systems with similar size and complexity.

**Credibility** concerns the confidence in the result. We adopted several threat mitigation strategies, such as sending interview transcripts to interviewee for correction; involving three researchers on study design and review, methodological triangulation and detailed
documented steps of data collection, processing and analysis. Thus we believe that the resulting detailed protocol serves as a good means to replicate this study.

4. Analysis of multiple platform development

The first iteration focused on describing state-of-practice multiple platform development. We explored the reasons that lead to diverged codebases (Section 4.1), the extent of divergence and parallel development among the codebases (Section 4.2), challenges with multiple platform developments (Section 4.3) and the extent of redundant work across codebases (Section 4.4). Figure 3 describes a conceptual scheme as a main qualitative finding. The frequency of topics across Spark projects are given in Appendix B.

4.1 What are the reasons for code divergence in Spark?

Avaya Spark-related project teams recognize the benefits of preserving a common codebase across platforms - i.e. not forking the code repository. Benefits include efficiency in development and maintenance because changes are only made once, and improved quality because validation and verification activities can focus on one version of the change instead of being distributed across many versions. SVN log history reveals a lot of forks that were created due to several reasons, such as for a new release, new features, a new hardware protocol, service packs, 3rd party patches, SV testing, incompatible schedules, and debugging. These fall into four main categories: schedule constraints, technical variation in the platform, coordination overhead in maintaining a single platform, and organizational differences.

Figure 3: Conceptual scheme of multiple platform software development
4.1.1. Feature difference

The first reason for forking a codebase is the significant changes that need to be implemented when a new feature is introduced. Spark system has many individual product teams. 80% of our interviewees mentioned that a fork occurs when a significant amount of feature variety was introduced to the project. For instance, two projects PrE and PrF implemented a set of signaling engine functionalities in iOS and Android accordingly. Early in its project stage, PrF stopped sharing the signaling engine codebase because of the specific feature introduced for iOS: “… after the first six months, the signaling engine stopped synchronization with project PrE. From that time, it was branched off from PrE and developed independently, due to technical differences. The current situation is that it has features that other iOS clients do not use” (P5).

Another typical example of forking due to feature variety is project PrG and project PrD implementing clients on Windows and iOS platforms. In the beginning there was a close collaboration between the two teams as User interface (UI) code was shared for the first two releases: “...the reason is that the UI team is the same for both Windows and iOS versions. The design and logic behind them is pretty similar…” (P8). The codebase for subsequent releases diverged, however, because of project differences in user interface requirements. Developers in PrG also mentioned that this issue made it difficult to collaborate among teams. The assessment of feature variety becomes a part of the forking decision-making process in PrA: “...the decision is usually made by trying to understand what are the main changes coming in from other releases. For example, if we are working for a PrA release and we know that a release from another project is going to introduce major change that needs to be updated in our release, then probably we want to branch out before that. This is a purely risk management activity.” (P9)

4.1.2. Schedule difference

The second reason for forking a codebase is the different requirement for release of the same feature in different platforms. 60% of interviewees mentioned that codebases get diverged due to different project releases. Typically, one project may require a stable code base with minimal, tightly controlled changes, while another project may be in active development with rapid code changes. For example, PrA shared some common features with PrB, and originally they were in the same codebase. Later in the project timeline, PrA reached a deployment milestone while PrB was still under development. However, the PrA team created separate code for the common feature to assure code stability: “the next delivery date of PrA is [Date X] while the coming software delivery date of PrB is later at [Date Y]. Therefore, while PrB is still in development phases with testing and debugging, PrA needs to be stable for delivery”. (P1). Besides, projects with different development schedules have also chosen to diverge their codebase. For example, we observed that code divergence occurred among projects with different numbers of sprints or with different sprint durations.

4.1.3. Coordination overload

The third reason for forking is the amount of coordination activities need to maintain development in the same codebase. 30% of interviewees mentioned that when two projects share the same code base, cross-project coordination is required to manage the impact of changes to the code base on each project. We observed that projects have chosen to fork their codebase when too much coordination is required. P6 stated: “…in general, there is no good
reason to have separate platforms of the same hardware. Historically there were separate teams working on codebase PrF. Later in the project, they had been communicating too much and (hence) the codebases got separated. Here is another example: “...the reason for stopping synchronization (between PrC and PrD) was too much overhead, no dedicated people to maintain the [shared] source code” (P8)

4.1.4. Organizational difference

The fourth reason that decides the structure of fork is the organizational structure. 30% of interviewees stated that code divergence between two projects with common features and common schedule occurred because these projects belonged to different organizational units. Management needed (or wanted) total control of resources if they were responsible for product delivery in their organization: “...it (coordination efforts) has been mostly on ipad and windows team, because we came from the same team and we were working very closely together on the same area . . . after that, we have been divided into different teams, as the managers were divided into the pure desk- top and windows team and the purely mobile teams ...the organization split in the different ways from ...” (P2)

In Spark, codebases are forked for outsource development teams. It is a common case that the outsourced team created its own version of the code base to establish code ownership of features and defect fixes: “...I find the best thing working with external teams, especially from different time-zone and locations is that if you have a really good clear process, especially with feature forking model, it is very easy for people to work together...they work on features and wrap it up, deliver back to mainstream in the end of release...” (P10). Strictly speaking, we consider feature work to be an instance of branching, not of forking. However, in this case the outsourced team used their own version control repository and worked on an entire sequence of releases, not on just on individual features.

4.2 How much parallel development is in Spark?

Literature reveals little information about how we should measure parallel development in multi-platform systems. We interviewed developers in Spark and collected three measures that could be useful: number of forks (NumFork), number of files under parallel change (NumFilePara), and fraction of files under parallel change (FracFilePara). NumFork and FracFilePara were cross-sectional measures and collected by the end of Apr-2014. NumFilePara was collected for every month from Jan-2009 to Apr-2014.

4.2.1. Number of forks (NumFork)

The first measure of parallel development is the number of forks each project relates to. A project often has code files being copied or copied (related files) from other platforms. A project with a high number of related forks is a project that has related files from many different places. Therefore, the number of related forks represents the extent of parallel development at the project level. Figure 4 presents a distribution of the number of forks versus project size. It shows that the number of forks varies among Spark projects from 1 to 200. The median number of forks of projects in the Spark family related to other projects is 33.

We consider fork relationship as an undirected graph. However, a codebase may have four related codebases, in which three of them are its forks and another is the original codebase that these forks derive from. We, therefore, examined codebases that have only a few related
codebases, because they can represent for a codebase renaming or code moving. There are 34 projects (79%) have code files spread over more than 10 forks. There are three projects with 70 or more forks, all of which are common engine components that deliver code to many different applications across many platforms.

4.2.2. Number of files under parallel change (NumFilePara)

The first measure cannot distinguish between a project with many related files and a project with few related files. Coordination needs may be different in a project that clones a single file from ten different platforms and a project that clones ten files from a single other project. To address that, we consider the number of files under parallel development. A file undergoes parallel development if there exists at least one related file that and both files are changed together over a specific period in time.

Figure 5 presents the number of files under parallel development from Jan-2009 to Apr-2014. On average, there are 2459 files under parallel development each month. The peak occurs in Oct-2009 with 68626 files being changed in parallel of which 66642 were from the advd_project. There may have been a large project reorganization at that time, involving code transfers and merge activities within advd. The most recent peak occurs in July-2012 with 16915 files under parallel development, of which 15203 files belonged to h32396x1 project.

4.2.3. Fraction of files under parallel change (FracParaFile)

To account for project size, we measure the portion of files under parallel change for each project. Figure 6 presents the percentage of files under parallel development in each project. In project selection phase, we filtered seven projects with 0% percent of files under parallel change. These projects represent code transfer from old to new codebases without any parallel development. The metric value ranges from 12.75% (project onexmobile/iOS) to 93.52% (project contextengine.git). The median value is 61%, showing that for a typical project, for every five files there are three files under parallel development or maintenance activities.
4.3 What are challenges when working across multiple platforms?

From the interview data we find four primary challenges developers often face with while working with diverged code bases. They are: growing coordination debt, redundant development effort, coordination with procedurally diverged team, and coordination across functional units.

4.3.1. Growing coordination debt

Multiple platform development might increase the needs of solving coordination debt across platforms. Coordination debt is extra effort to coordinate development activities when a cross-platform piece of code is implemented in the short run (e.g. to deliver a product release). Later the code needs to be updated for a long-term solution (e.g. to deliver subsequent releases or to share code among similar products), which requires coordination effort among relevant platforms. A defect or new feature often needs to be implemented in different versions across different platforms. The changes in one platform that need to be applied in another platform are often delayed or not implemented.
The reason for the delay might be the lack of people to resolve these issues. Also, the priority of issues may vary among different platforms. Platforms where the issues are of high priority are the first ones to implement a fix, while other platforms may not want to fix the issue at all. Integrating source code from different platforms also creates extra technical debt due to merge conflict, as mentioned by P10: “We migrate software from platform to platform, or team to team, and this builds up technical debt. We need to continuously invest in software — it is getting old even if you don’t touch it”. Spark system has many individual product teams. Because of overall staff constraints, each product team is thinly staffed and yet is responsible for the entire product. As a result, schedules are tight with little time to reduce the MR backlog: “Chasing a deadline created technical debt. Technical debt needs to be caught and dealt with earlier.” (P8).

The more development with quick-fix components happens, the more coordination debt is piled up. In several projects, multi-platform development comes with a rapidly increasing MR backlog, creating pressure on developers, which leads to “work-around” MR resolutions. For instance, P5 stated that, “The project has an ever growing backlog. It has grown to [figure] Severity level 1/2/3 issues for all the releases… Many of these are duplicates. If it were not for the urgency of shipping 6.2 the backlog would be addressed. We have stability issues we do not know the cause of”.

4.3.2. Redundant development effort

Multiple platform development introduces redundant work in both implementation and testing activities. First, Spark system contains a lot of legacy code, and a lot of builds contain duplicated files. Product teams who are responsible for one Spark platform were often unaware of many changes in other Spark platforms. Consequently, features and defect fixes are implemented independently for each platform, resulting in development that is unintentionally duplicated. For instance, P3 stated that: “Some files included in the root are present multiple times in different codebases. For example some files are generated by multiple packages. This is redundant but also makes it difficult to know for sure which version of the file ends up on which phone release…” The other mentioned that: “…it is important to make sure that you identify the correct source for the file as found on the latest phone release today and remove the other ones. It is more difficult to check across different platforms.”

Second, diverged codebases induce redundant test effort because each platform’s test team independently tests similar features or functionalities from different platforms. Unless they know other teams’ test cases of similar features, testers independently create their own test cases as shown in this example: “the PrA team now only tests for PrA code and PrB team only tests PrB code. If there is a cross delivery, both teams have to test other product as well. Hence, both test teams get hit by the same amount.” (P2). Another example: “when a defect involves both Broadcom hardware and application products, both sides need to be able to reproduce the defect to file the main root-cause” (P8)

4.3.3. Coordination in globally distributed teams.

In a multi-platform system, coordination among global development units is more problematic. Under the same organization, there is still a process dispersion occur among development teams. For instance, one team adopted SCRUM with many small sprints while another team used ad hoc and informal agile development, without splitting tasks into sprints and continuously receiving and developing tasks: “DSP Team in Sweden hadn’t work with a
Scrum environment before. It took 3 sprints to get them in the right mindset and another few sprints for them to be fluent in the process”. The different development approaches hindered in-depth discussion of technical tasks at team level. Development infrastructure, such as development tools and test environment also differed across geographical distance and made it difficult for inter team collaboration: “Between the Sweden and US codebase there is a difference in the way they operated. Reproducing the failure in different locations is difficult. To make sure everyone keep the infrastructure up to date and all have a reference point. Sweden runs a slightly different version from the US one”. (P3).

The coordination procedure also needs to be adapted for time zone difference: “Even in scheduling meetings (and offshore are accommodating), it is hard, and so we need special meetings, and that provides a drag. On paper some things make sense, but in practice they do not.” (P9) Talking to developers gave the impression that duplicated development effort often occurs on the border of software and hardware teams. Furthermore, components from outsource teams or 3rd party suppliers are expected to have a longer collaboration time while waiting for their feedbacks. These waiting times blocked development activity and delayed code integration across platforms.

Coordination across external organization boundary, i.e. with 3rd party, is more problematic. P1 talked about collaboration between PrA and PrB: “We found a defect in PrB codebase around the Broadcom hardware. Broadcom provided a fix ... and also introduced patches for that. We were working with Broadcom on PrA and would propagate the fix to PrB. ... We were not in the position to take the patch in the current release of PrB. This happened a couple of times and then we had different patches from Broadcom on PrA and PrB. The main challenge in consolidating two branches was matching the Broadcom patches”. A particular and often very important case of redundant testing is testing third party components. “When a defect involves both [third party vendor] hardware and application products, both sides need to be able to reproduce the defect to find the main root-cause.” (P6)

4.3.4. Coordination across functionally diverged teams:

With diverged code bases, each project has its own priority of features and defect fixes. As a result of a continuous evolution, the whole product family becomes extremely complex. Consider, for example, that the likely complexity of the diverged code base leads to few, if any, engineers who understand in detail this entire code base. Individual development teams typically do not have “who does what ” and “who knows what ” knowledge about other teams as shown with this example: “Once you move outside the core, you have different platforms, schedules, and product managers, etc. In many cases working with another project is like working with a different company. There are different things to manage, so the orchestration of it all becomes problematic.” (P5).

We found that duplicated testing effort is more critical in high-level testing teams, such as with the PV team and SV team. At this level, testers did not have insight about releases and how they related across platforms. A developer stated, “This is something outside of our hand. What we can do is to tell the SV team that these products should be tested, there is a new release with these patches, we need you to pick it up and run with it ... We do that by writing in the release notes. . . . But usually the SV team organizes themselves to secure resources for program vs. testing these patches just like we do in development.” (P8)
4.4 How much redundant work in Spark?

Software Engineering literature investigated several measures of code clones at code level. However, we did not find any study investigating the duplicated work at file and MR level. Therefore, we introduce two measures to address the extent redundant work at commit and MR level.

4.4.1. Number of redundant commits (NumRCom)

Figure 7 illustrates the concepts of redundant commit and redundant MR. There are three projects Proj1, Proj2, and Proj3 with some code files inside. There are four MRs associated with different files as shown in Figure 7. File F4 clones file F2 and continue development with 3 commits. A change in a file, for instance, F2 in Project 2, can be associated to a specific MR, for example MR1. There is a cloning relationship among MRs, for instance, between MR2 and MR3.

NumRCom is defined as a sum of the fraction of redundant commits over all files of a project. Two commits in a redundancy relationship if they belong to two related files. We use the following reasoning to justify the label “redundant”: a commit to a file that was cloned from an original file is likely related to the original one, i.e., defect fixing or code refactoring. A defect-fix commit in one project may need an independent implementation in another project where developers may not be aware of the original fix. Such an implementation would be clearly redundant because, if aware of the original fix, the second team may be able to reuse it in their project if the code has not diverged too much. Therefore, NumRCom may reflect the amount of redundant development work.

In the example, number of redundant commits in Project 2 is the sum of the fraction of three commits in F2 for two other related files and the fraction of two commits in F3 for another related file:

\[
\text{NumRCom(Proj 2)} = 3 \times \frac{2}{3} + 2 \times \frac{1}{2} = 3
\]

4.4.2. Number of redundant MRs (NumRMr)

NumRMRs is defined as a sum of the fraction of redundant MRs over all files of a project. We considered two MRs in a redundancy relationship if they are associated with files that are related. In the example in Figure 7, MR1 is redundant because F2 is related to F4, which, in
turn, is associated with MR3. Redundant MRs do not represent MR clone relationships (for instance, a relationship MR2-MR3 is cloning relationship and is different from redundancy of MR1-MR3). We use the following reasoning to justify the label “redundant”: If there are two related files each with a distinct MR, then it is likely that these two MRs are related. A high number of redundant MRs implies a large number of related tasks that are implemented independently in several projects.

In the example, number of redundant commits in Project 2 is the sum of the fraction of a MR with F2 for two other related files and the fraction of a MR with F3 for another related file:

\[
\text{NumRMR(Project 2)} = 1 \times \frac{2}{3} + 1 \times \frac{1}{2} = 1.167
\]

Figure 8 presents number of redundant MRs in Spark. The number of redundant MRs ranges from 10 (dca/onexc_library/) to 45109.5 (h32396x1/). The median value is 748.75 redundant MRs per project.

Figure 8: Number of redundant MRs

5. Analysis of coordination in parallel development

5.1. What are effective coordination mechanisms?

During interviews, there were many coordination mechanisms mentioned, such as: automatic detection of build failure, continuous builds, dashboard, wiki, JIRA and mailing list. While these mechanisms have been extensively mentioned in SE literature, we would like to focus on three approaches that are especially interesting in the context of multi platform systems. Many projects deployed at least two of these approaches: cloned MRs across platforms, a cross platform coordinator role, and cross-project MR review meetings.

5.1.1. MR cloning

Cloned MRs across platforms, where a cloned MR is a replica of the original MR, containing the same information stored in the original issue, such as Summary, Affects Versions and Components. The cloned MR is typically linked to the original MR. Formal defect tracking
systems are used as a standard practice in all Spark-related development teams. To coordinate a common defect fix among projects, many projects create a cloned MR as described by P7: “...when there is a platform MR ticket linked to an application MR ticket that requires resolution, the ticket in the application would be assigned to a developer who is responsible to track the changes. He would do so through the relevant focal point in our team to the platform team. Once the platform ticket is resolved, the person assigned should verify this, add the info to the ticket and sync up with the assigned build person on the revision the needs to be committed. The assignee would update the ticket with the changes done (move it to resolved with relevant info and create clones if required).” (Wiki page PrH, P7).

5.1.2. Cross-platform spanner

Technical staff members who address cross-platform issues are critical to maintaining knowledge flow among projects. We identified two types of spanners, formally assigned spanners and emerged ones. Formal spanners might be a designated team consisting of developers from different Spark projects, such as SIP, H323 and 1XC, resolving issues of common concern, such as changes in tools or process, buyback planning, and dependencies between projects. Other projects assigned a developer responsible for examining all components being developed for some other platforms. As an example, an appointed developer made the following observation: “...there is a Spark engine team who look on both sides, Windows and iOS, to check for a defect across the platform . . .” (P8).

Emerged spanners are developers that work on many platforms and hence gained a lot of cross platform knowledge. These developers are not formally assigned to any coordination activity, but often contribute much for identifying, communicating and resolving defects across platforms. These developers can be automatically identified from a social network analysis of developers and codebases.

5.1.3. Cross-platform MR review meeting

The MR review meeting is an important standard practice in all Spark related projects as shown by this typical example: “every week the scrum team should meet for about an hour to review the product backlog. This is the process of estimating the existing backlog using effort/points, refining the acceptance criteria for individual stories, and breaking larger stories into smaller stories” (Wiki page – Project PrK).

<table>
<thead>
<tr>
<th>Goal</th>
<th>Questions</th>
<th>Metrics</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantitatively describe state-of-practice development and coordination of a multiple platform system</td>
<td>How much parallel development in Spark?</td>
<td>Number of forks (NumFork)</td>
<td>4.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of files under parallel change (NumFilePara)</td>
<td>4.2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fraction of files under parallel change (FracParaFile)</td>
<td>4.2.3</td>
</tr>
<tr>
<td></td>
<td>How much redundant work in Spark?</td>
<td>Number of redundant commits (NumRCom)</td>
<td>4.4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of redundant MRs (NumRMr)</td>
<td>4.4.2</td>
</tr>
<tr>
<td></td>
<td>How much coordination effort has been spent?</td>
<td>Fraction of Cloned MRs (FracClonedMR)</td>
<td>5.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cross-platform spanner (NumSpanner)</td>
<td>5.2.2</td>
</tr>
</tbody>
</table>
Cross-project MR review meetings include project managers, technical managers and lead developers from each participating project. At each meeting, participants review requests from each participating team, review and create tickets, and make sure team members are aware of what is coming to them. “…every time there is an open code inspection everyone in the team can participate …we have a mandatory of minimum inspection from two reviewers when submitting our code…” (P10)

Cross-project MR meetings help create a culture of regular informal communication to clarify conflict issues, cloning issues and other issues as described in this example: “synchronization is done by talking to the other team, such as the platform team. In the development level we are discussing everything, such as the testing environment. For the Tel Aviv site, every time they put in a fix we need to test whether it introduces a defect to the application. Overall, there is a lot of coordination that happens on a daily basis…” (P4)

5.2. How much coordination effort has been spent?

We derived metrics to quantify the amount of coordination via two main coordination mechanisms, which are cloned MRs and cross platform coordinators.

5.2.1. Fraction of Cloned MRs (FracClonedMR)

As noted from our qualitative analysis, cloned MRs are an important inter-project coordination mechanism: “We take a ticket from one platform branch, cloning them to another platform branch and linking them to the application feature. Basically it helps us to track back the issues” (P7). Initially, we came up with a measure of the number of cloned MRs of a project. A project can clone from another one, leading to all MRs cloned from one. A MR cloned to many projects indicates more coordination effort than a MR cloned to one project. Besides, there might be only one developer who keeps tracks of many different cloned MRs.

However, a large number of cloned MRs might be due to the large number of MRs that a project has. Therefore, to control for size, we used a ratio of the total number of MRs divided by the total number of cloned MRs in a project (defined above), or fraction of cloned MRs. A large fraction of cloned MRs from other projects implies a high number of coordination efforts to clarify a repeated issues or applicable defect fixes across different platforms. Figure 9 presents the fraction of cloned MRs over number of code associated MRs in each project. The figure ranges from 0% (project cs1kgps/) to 75% (project ngue_flareandroid.git/). The median value is 26.8%, showing that for a typical project, in every four code associated MR, there is one cloned MR.

5.2.2. Cross-platform spanner (PortSpanner)

Qualitative findings reveal an emerged type of cross platform spanner. An emerged cross platform spanner is a person who has knowledge of a large number of codebases. We operationalize that by identifying developers who contributed code to multiple codebases.
The activity of emerged cross platform spanners may be considered as the measure for the amount of coordination that is done in the project. To control for project size, we use a ratio of the cross platform spanners and total number of developers who submitted code to a platform. Table 5 presents the cross platform spanners for several projects. The table shows the minimum, median, and maximum value for the number of spanners who contributed code to a certain number of codebases. A large number of developers participated in multiple codebases, for instance, a median (90.5%) number of developers participated in more than two projects. One project in Spark has 9.4% number of developers who contributed to more than 30 codebases.

It is common for a developer to work on many different codebases during their tenure at Avaya. For example, 100% of developers in Spark projects have participated in at least two projects (Column 2 – Table 5). To determine who is an emerged boundary spanner that has significant knowledge about multiple platforms, we select a developer that submits changes to at least 1/3 total number of forks for the project. As a median Spark project has 33 forks, we used a threshold value of 10, which means that a developer is considered to be a boundary spanner if she submitted code to more than 10 codebases. Figure 10 shows the portion of boundary spanners among Spark projects. For instance, in a median project there is one boundary spanner for every four developers. The maximum fraction of boundary spanners in a project is 75%, which is the case of dca/oneexc_library.

### Table 5: Statistics of project spanners

<table>
<thead>
<tr>
<th># spanned proj.</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min (%)</td>
<td>16.4</td>
<td>12.6</td>
<td>11.1</td>
<td>9.5</td>
<td>8.6</td>
<td>4.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Quarter 1st (%)</td>
<td>72.6</td>
<td>47.5</td>
<td>42.6</td>
<td>35.05</td>
<td>25.65</td>
<td>16.55</td>
<td>1.95</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>Median (%)</td>
<td>90.5</td>
<td>78.6</td>
<td>72.1</td>
<td>67.3</td>
<td>55</td>
<td>25.4</td>
<td>5.2</td>
<td>1.9</td>
<td>0</td>
</tr>
<tr>
<td>Quarter 3rd (%)</td>
<td>95.85</td>
<td>91.15</td>
<td>84.2</td>
<td>78.1</td>
<td>68.95</td>
<td>42.75</td>
<td>13.3</td>
<td>3.6</td>
<td>0</td>
</tr>
<tr>
<td>Max (%)</td>
<td>100</td>
<td>98.6</td>
<td>95.9</td>
<td>93.2</td>
<td>88.1</td>
<td>75</td>
<td>66.7</td>
<td>9.4</td>
<td>0</td>
</tr>
</tbody>
</table>
5.3. Evaluation of coordination mechanisms

Qualitative findings suggest cross-platform boundary spanners and the MR cloning as two coordination mechanisms that were adopted in many projects. This section evaluates the effectiveness of these mechanisms by modeling their relationship to customer found defects (CFD). CFDs are carefully tracked in Avaya so we can identify whether or not an MR is related to a CFD. More specifically, as customer reported defects, we only consider MRs of medium and high severity that were determined to be software defects and that have been a result of a customer reported issue.

MR cloning is a feature of the issue tracking system and is available in all Spark projects. Before submitting a new MR, developers should attempt to determine if the MR has been reported elsewhere. Based on the feedback from developers it is reasonable to assume that the MR cloning feature tends to be used when possible. The extent of MR cloning adoption may imply the technical-dependency-induced coordination needs (Cataldo, Wagstrom et al. 2006; Cataldo, Herbsleb, et al. 2008). Many cloned MRs may be observed if a large part of the codebase overlaps with other projects/platforms. Using cloned MRs, development work that is done in one project will need to be coordinated and integrated (by merging changes) into another project later. Therefore the number of cloned MR represents the extent of coordination needed among these projects for resolving duplicated development work. A project that has many cloned MRs, consequently, also has significant coordination needs, which are deferred. It has been observed that coordination breakdown occurs when coordination needs are not satisfied, leading to a lower software quality (Cataldo et al. 2009). The large number of cloned MRs may be due to the large amount of MRs that a project has. To control project size, we use: FracClonedMR = NumClonedMR/ NumMR. Based on the considerations layed out above, we propose the first hypothesis:

\[ H1: A \text{ project with a larger fraction of cloned MRs has more CFDs.} \]

Boundary spanner is an essential role that might be explicitly assigned or implicitly emerge in a software project. Every project will have an individual or a group of people taking responsibility to connect with external stakeholders by, for example, transferring and navigating task and team status information. In Spark projects, we identified the implicit type of boundary spanners, who have contributed code to more than 10 platforms.
On one hand, the number of boundary spanners might imply the availability of knowledge about different platforms in a project. For a project with a high number of spanners, coordination requirement are more likely to be noticed and addressed and, consequently, the chances of customer reported defects would be decreased.

On the other hand, the implicit boundary spanners’ time and effort are spread through many projects. The high number of spanners might also imply that the projects do not have sufficient dedicated resources and, therefore, leads to more customer reported defects. The number of spanners tends to increase with the number of active developers in a project. Common folklore in software engineering asserts that “too many cooks spoil the broth” when it comes to code development and maintenance. Literature suggested a positive association between the number of developers and the number of software defects (Anbalagan and Vouk 2009, Bettenburg and Hassan 2010, Pinzger, Nagappan, et al. 2008).

A large number of spanners may be due to the large amount of developers contributing to the project. To control for project size, we used: \[ \text{PortSpanner} = \frac{\text{NumSpanner}}{\text{NumDeveloper}}. \] An implication from a large portion of boundary spanners is that instead of availability of multiple-platform knowledge in a project, there might also be a high extent of knowledge dispersion and lack of dedicated development efforts in the project. The dispersion of people and expertise across global teams was reported to be associated with team performance and software quality (Ramasubbu et al. 2011). Similarly, there might be a configuration of boundary spanners and focused developers that leads to positive project outcomes. If the portion of boundary spanners becomes too large, it might negatively impact software quality.

Based on the considerations given above, we propose the second hypothesis:

**H2: A project with a larger fraction of boundary spanners has more CFDs.**

Logistic regression (LR), sometimes called binomial logistic regression, is a widely used statistical technique to model probability where the dependent variable is either zero or one. It is used to predict the probability of a particular outcome, given a particular set of circumstances.

LR does not require some of the assumptions of multiple linear regression, such as the linearity between the independent and dependent variables, normally distributed variables, or homoscedasticity [26]. One assumption of a logistic regression model is that all observations should be statistically independent. We regard the labeling of an MR as a customer related defect observation of 1 and the remaining defect MRs as 0.

\[
\pi(Y = y|X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \frac{e^{x + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n}}{1 + e^{x + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n}}
\]

We used a natural logarithm transformation for all predictors shown in Table 6. This helps reduce the impact of highly skewed distribution of the collected metrics. The 25869 MRs from 43 projects represent our observations. Due to the small size of the sample, we could not use many variables in our models. Therefore, we included two variables represented for the coordination mechanisms and also two size measures, as shown in Figure 11.

All estimated coefficients are statistically significant (different from zero). Both measures project size are positively associated with the probability that a defect is a CFD. This implies
that the project with more files and more developers has more chances that a defect will be reported by a customer.

FracClonedMR has a positive coefficient, showing that in a project that has a large portion of cloned MR also has a larger fraction of CFDs. We therefore, do not reject hypothesis H1. This implies that projects with high fraction of cloned MRs also have a high fraction of CFDs. The finding seems to support the argument that a project with large adoption of cloned MRs is also a project with large amount coordination needs, justifying the fraction of cloned MRs as a proxy for coordination needs. The effect size calculated by odds ratio of median values and third quartile of the FracClonedMR shows that an MR is a CFD increase 36% if we go from project with a median fraction to a project that has a third quartile of the fraction of cloned MRs.

FracSpanner has a positive coefficient as well, indicating that in a project that has a large fraction of boundary spanners also has high fraction of CFDs. Therefore, we do not reject hypothesis H2. The finding suggests that a large portion of boundary spanner is not an optimal configuration for a project to achieve good customer-driven software quality.

6. Discussion

6.1. Summary of findings

We summarize challenges and practices of developing multiple platform systems in this section.

- **What are the reasons for code divergence in Spark?** We found four main reasons for code being diverged, namely technical variation in a platform, schedule constraints, coordination overhead in maintaining a single platform and organizational differences. The decision of forking codebases comes from not only technical requirements but also managerial issues.

- **What are coordination mechanisms using?** We found three main mechanisms for coordination across platforms, which are cloned MRs, a cross platform coordinator role, and MR review meetings. These approaches relied on developers with knowledge of different platforms and procedures and infrastructures that enables knowledge sharing.

- **What are challenges when working across multiple platforms?** We identified four major challenges, namely growing technical debts, redundant development effort, coordination with globally diverged teams, and coordination across functional units. Developers often find problems in working across the organizational boundary between projects, functional groups and locating relevant code as well as developers.

- **How much parallel development in Spark?** On average, a Spark codebase has 33 related forks. 97% of projects in the Spark family have code related to other projects. A typical Spark codebases has 61% of files under parallel development.

- **How much coordination effort has been spent?** On average, 26.8% MRs are cloned MRs. In a median Spark project, number of cross platform spanners constitute of 25% total number of developers.
<table>
<thead>
<tr>
<th></th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>Pr(&gt;</th>
<th>Dev. Epl.</th>
<th>Eff. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>-7.75</td>
<td>0.2</td>
<td>0.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>log(NumFile + 1)</td>
<td>0.51</td>
<td>0.141</td>
<td>0.00</td>
<td>546.65</td>
<td>1.5</td>
</tr>
<tr>
<td>log(NumDeveloper + 1)</td>
<td>0.37</td>
<td>0.02</td>
<td>0.00</td>
<td>15.86</td>
<td>1.46</td>
</tr>
<tr>
<td>log(NumSpanner/NumDeveloper + 1)</td>
<td>2.38</td>
<td>0.19</td>
<td>0.00</td>
<td>43.02</td>
<td>1.36</td>
</tr>
<tr>
<td>log(NumClonedMR/NumMR + 1)</td>
<td>5.64</td>
<td>0.19</td>
<td>0.00</td>
<td>971.91</td>
<td>2.21</td>
</tr>
</tbody>
</table>

Model summary: pCFD~log(NumFile+1)+log(NumDeveloper+1)+log(frSp+1)+log(frCl+1).
43 projects, 25869 MRs. Deviance explained 0.4

Qualitative and quantitative data suggests several recommendations for coordination and maintenance of multiple platform systems. Section 6.2 and 6.3 presents the recommendations, which are summarized in Table 7.

### 6.2. Towards merging of diverged codebases

Our qualitative findings revealed several development challenges in Spark. Two current solutions to handle diverged codebases are to either merge code into a common codebase or to coordinate development activities across codebases.

**Table 7: Recommendation to manage multiple platform development**

<table>
<thead>
<tr>
<th>Area</th>
<th>Question</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supporting merge of diverged codebases</strong></td>
<td>What determines the time to merge codebases?</td>
<td>Consider release schedule and boundary spanner’s availability</td>
</tr>
<tr>
<td></td>
<td>How to merge diverged codebases?</td>
<td>Fix short-term defect before merging</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Determine policy of vertical and horizontal code integration across platforms</td>
</tr>
<tr>
<td></td>
<td>Which parts of the codebases should be merged?</td>
<td>Merge components that have a lot of boundary spanners</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Merge components that have a lot of cloned MRs</td>
</tr>
<tr>
<td><strong>Supporting maintenance of diverged codebases</strong></td>
<td>How are codebase forks should be structured?</td>
<td>Adopt architectural-driven fork in intra-project context</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adopt organizational-driven fork in inter-project context</td>
</tr>
<tr>
<td></td>
<td>How to coordinate among diverged codebases?</td>
<td>Focus of cross-platform coordination effort on the files under parallel development</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Establish formal notification processes for cloned MRs across codebases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enhance communication of redundant work between testing team and development team</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Establish integration points among globally distributed development teams</td>
</tr>
<tr>
<td></td>
<td>How to deal with redundant work?</td>
<td>Replace short-term defect fixes with a long-term resolution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Establish united code integration policy across platforms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revise the variation of branch naming conventions</td>
</tr>
</tbody>
</table>
6.2.1. What determines the time to merge codebases?

The first concern is whether a codebase should be merged or maintained separately. Several factors should be taken into account when deciding merge strategy, namely release schedule and boundary spanners’ availability:

- Release schedule: typically, merging diverged codebases occurs in a context that different codebases have their own on-going schedules and deadlines. Hence the first issue is whether there is a specific time period when all involved codebases are ready for merging. Our qualitative data reveals that diverged codebases should be merged at the end of development iteration of each platform, or at the time when both codebases are stable.

- Boundary spanner’s availability: a merge challenge is the lack of developers with knowledge of the whole system. In quantitative analysis, we revealed that a there is a significant number of developers who contributed to multiple codebases. Therefore, it is important to compose a team of relevant developers. Moreover, the team should have ability to deliver and support this parallel development and meet the deadlines.

6.2.2. How to merge diverged codebases?

Codebases can be merged from different development branches to the main branch of a platform (horizontal merge). Code can also be merged from two development branches from different platforms (vertical merge). To resolve merge conflict in the vertical approach, the team needs to decide which changes from which branches to take in first. A vertical merge has to consider the integration direction. We found several MRs opened for forward integration (merge code from development branch toward main branch) and reverse integration (merge code from main branch to development branch).

It is necessary to have guidelines for merging across platforms, for example, in which circumstances should reverse integration be done before/after forward integration. At the current state, we observed no consistent integration policies across projects. The integration is done in a mixed pattern: 1) at the beginning and the end of a sprint for development branch, 2) at the beginning and the end of a release for release branches and 3) at the beginning and the end of a project. Merging between two development branches from different platforms and skipping update to the mainstream creates some MR issues in both platforms.

6.2.3. Which parts of the codebases should be merged?

When the decision on codebase merge is made, a next step is to identify the components for merging. There are two type of components that merging can help reduce coordination and extra development efforts:

- Merging components on projects with a large percentage of files under parallel development to improve software quality. The previous recommendation suggests to selectively coordinate files under parallel development. An alternative solution to handle diverged codebases is to provide a unified codebase for parts of projects with large fraction of files under parallel development. This approach has been adopted in Avaya, though its effectiveness has not been investigated yet.

- Merging components on projects that have a lot of common coordinators to reduce duplicated development effort. These components have many related files that are
worked by the same developers. This reduces redundant work and coordination effort to propagate information on these changes to other developers.

6.2.4. Implication for tools
When merging diverged codebases, our results reveals an implication for supporting identification of files under parallel development and the associated projects. We found that there is a significant number of files under parallel development. There are also a lot of related files that are changed at the same time without awareness of their owners. A tool that identifies all related files under parallel development for specific projects will help developers to reduce the possible redundant works and prevent integration conflicts in later phases.

6.3. Towards maintaining diverged codebases

6.3.1. How should codebase forks be structured?
In the case of maintaining separate codebases, the issue is whether to maintain forks according to architecture structure or organizational structure (Shihab et al. 2012). One strategy to create diverged codebase is to dedicate one fork per component. Doing so allows software components to be developed in isolation. However, in certain cases, multiple components are modified in a single fork, causing forks to cut across the architecture. Another strategy is to create separate forks for each organizational team. This approach allows better teamwork, code ownership and management of local teams. Our finding suggests that architectural driven forks are adopted in an intra-project context while organizational driven forks are adopted in an inter-project context.

Maintaining multiple diverged codebases implies a significant amount of redundant work. Coordination effort for development of overlapping parts of codebases might lead to overload (and a fork) when product backlog increases. Forking can be used as a mechanism to defer the coordination needs of non-significant development work. Fork can be created according to organizational structure, release schedule or architectural modules.

6.3.2. How to coordinate among diverged codebases?
When a decision on maintaining diverged codebases is made, it is important to have an effective coordination approaches across codebases. Our result suggests four activities that help to improve coordination:

- Focus cross-platform coordination effort on the files under parallel development. It is necessary to identify a cross-platform file owner for each file under parallel development. Changes to a file that relate to many different platforms need to be propagated to relevant developers in these platforms.

- Establish a formal notification process for cloned MRs across codebases, such as periodic reviews of cross-platform issues. We revealed that cloned MR is an effective mechanism to coordinate defect fixes and new feature implementation across platforms. In the current state, development teams in Avaya manually create cloned MRs. In addition, cloned MRs are not systematically adopted through all platform teams. Consequently, it is beneficial to embed identification of cloned MR in formal review meetings at both intra platform and inter platform level. Moreover, there should be rules to handle cloned MR with different priority and severity levels.
• Enhance communication of redundant work between testing team and development team to reduce the number of duplicated test cases. Our qualitative finding suggested that high-level testing teams are not sufficiently aware of common features across different platforms. Duplicate test cases can be reduced by better communication of related files and cloned MRs to test teams. An example is to deploy the test environment together with the main codebase since a defect can occur in different operating contexts (application vs. factory test), and different phases of operation: boot, initialization, upgrade. The information needed to reproduce the defect should be transparent to both teams.

• Establish integration points among globally distributed teams. Another issue is found when coordinating tasks between teams with different development processes, i.e. scrum team vs. water fall team, teams with different scrum iteration duration, teams in different time zones and geographical locations. Many parts of Spark have uniform processes, but there is always tailoring at the project level. Projects mostly attempt to follow guideline at organizational level. While local processes and practices might be difficult to change in a short time, development of multiple platform codebases seems to require a better synchronization of development activities among these teams.

6.3.3. How to deal with redundant work?
Our findings also offer three recommendations to deal with redundant work:

• Replace short-term defect fixes with a long-term resolution. We found that a lot of defects have work-around solution and were temporarily closed. These defects might later either be reopened to have a long term resolution, derived to a new MR, or abandoned. For the defect that might be applicable to many different platforms, technical debt is created unless a long-term resolution is introduced as soon as possible.

• Establish a unified code integration policy across platforms. Code integration leads to amount of test and retest of old MRs. An integration policy can help to reduce amount of retest in relevant platforms.

• Review the variation of branch naming conventions and suggestions for good branch naming practices. Reduction of inconsistency in a global naming system helps to facilitate detection of duplicated work across platforms.

6.3.4. Implication for tools
When maintaining diverged codebases, we found that it is helpful to automate the process of identifying common coordinators and cloned MR across platforms:

• Automatically identify a set of common coordinators across platforms. We found that coordinators are an important source for knowledge across different platforms. For a specific developer, there is a set of coordinator that worked on related files with him. A tool that identifies a set of common coordinators across platforms will help to raise awareness of relevant developers.

• Support of identification of cloned MR across platforms. We found that cloned MR is an important coordination mechanism to maintain workflow across codebases. MRs that are associated to related files can be categorized in a group of potential cloned MR. Developers can review this group to identify the actual cloned MRs.
• Identify files under parallel development. For example, for any selected file in a code repository, identify the files in other branches or other repositories that are undergoing parallel development.

6.4. Effect of coordination mechanism

Kotlarsky et al. defined an integrative framework with four types of coordination mechanisms: coordination by organization design, work-based coordination, representations of work-in-progress, inter-personal coordination and technology-based coordination (Kotlarsky et al. 2008). Our qualitative finding reveals three types of mechanisms: consistent usage of collaboration tools across platforms (Cloned MR in Jira), organization design (cross-platform coordinators) and work-based coordination (review meetings). Organization design and work-based mechanisms play a role in shaping process and practices in adopting technology-based coordination mechanisms.

Quantitative result confirms the viability of two coordination mechanisms. Cloned MR is a mean to propagate information of redundant work across platforms. While cloned MR is the main approach to identify and trace duplicated defects, it is conducted in a rather ad hoc manner. Identification of cloned MR bases on individuals who have knowledge of different platforms. These people are also boundary spanners and also the main contributors of Spark. Maintaining such common gatekeepers for all forks of a project is necessary to manage coordination in the project.

Mismatches between coordination requirements derived and coordination behavior is shown to have a negative impact on the quality of complex systems such as airplane engines (Sosa, Eppinger, et al. 2004) or automobiles (Gokpinar, Hopp, et al. 2010). Satisfaction of coordination requirements implies a sufficient effort in coordination activities and availability of coordinators. Given that a technical dependency is identified, if there is not enough communication to solve the dependency, coordination requirement is not satisfied. Conversely, if coordination needs is identified, but there is no available developer to resolve the need, coordination requirement is also not satisfied. Our finding suggests that cloned MR mechanism and availability of boundary spanners adopted in projects with high defect-proneness. Combination of proper control of duplicated issues and effort of boundary spanner would help to resolve problems of resolving duplicated work.

7. Conclusions

Our study presents initial understanding on the state-of-the-practice of multiple platform development. The Spark family has a large amount of parallel development and a lot of redundant work. While intra-project branching is driven by technology, we found that inter-project forking is driven by market and management strategies. Our study also reveals cloned MRs is an important coordination mechanism in the Spark family.

We reveal some opportunities for future research in this area. In this study, we only investigated the extent of forking activities and redundant work via commits and MRs. In future work, we would like to address code divergence and assess the number of identical files between them in a more detail manner. In particular, we would like to investigate if there is any difference between a file that diverges much compared to a file that does not in term of
software quality and coordination efforts. Besides, future work can assess the types of governance (e.g. ownership, naming conventions, branching decisions,...) and its impact on the fraction and divergence of shared code. Last but not least, our study revealed the main reasons and consequences of code divergence. Further studies can investigate criteria used to decide whether or not to fork a code repository. The current status of the project, coordination cost and amount of redundant work would be important inputs for a decision making support tool on whether and when to fork a code repository.

References


Hochmiller, E. (2011). The requirements engineer as a liaison officer in agile software development. In Proceedings of Workshop on Agile Requirements Engineering, Lancaster, United Kingdom


Appendix A: Interview guide

Warm-up

1. Can you describe a bit about the history of the project?
2. What are the relevant projects and repositories?

Main questions

Section 1 - Code base divergence – current states and challenges

1. What are the reasons for this divergence (i.e. variation in new product, specific focus, too much forking, limited cross project coordination, different priorities)?
2. What are the main advantages of taking the code from other repositories? (cost saving, consistency, better architecture, …)
3. What are the main disadvantages of taking the code from other repositories? (coordination effort, complexity, …)
4. What are the challenges on working with developers from other organizational team (Sweden, India, Isarel)?
5. What are challenges with reusing test cases & infrastructure?

Section 2 - Merging – effective coordination approaches

6. What are the organizational policy that support/ limit the cross project collaboration?
7. How can a common defect with 96x1 is identified, keep tracked and resolved?
8. Which tasks/ project phases required close coordination among teams?
9. How test activities are coordinated?
10. How coordination with hardware team are done?
11. How a decision on new branch is made?

Concluding

12. Do you think any improvement on the policy/ infrastructure would help?
13. Any other reflection on team process?
Appendix B: Concept scheme

List of main qualitative concepts

<table>
<thead>
<tr>
<th>Topic</th>
<th>B189</th>
<th>Phone X6</th>
<th>H323 96x1</th>
<th>Flare Android</th>
<th>Flare iOS</th>
<th>Flare ipad</th>
<th>1XC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason for divergence</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Branching/ merging practices</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordination on testing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Coordination on defect fix</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Issues of distributed team, practices, infrastructure</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organizational policy, strategy</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Collaboration with engine team</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Collaboration 3rd party component</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaboration with Avaya component</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Collaboration with outsourced teams</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Collaboration with SV team</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Collaboration with other projects</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix C: Quantitative data

<table>
<thead>
<tr>
<th>Dur.</th>
<th>NoFile</th>
<th>NoPara</th>
<th>NoCloned</th>
<th>NoChange</th>
<th>NoRedundant</th>
<th>NoRedundant</th>
<th>NoMR</th>
<th>NoSpanner</th>
<th>Cus. defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1770</td>
<td>46954</td>
<td>33701</td>
<td>132</td>
<td>27</td>
<td>14212</td>
<td>210846</td>
<td>493</td>
<td>90</td>
<td>389</td>
</tr>
<tr>
<td>1649</td>
<td>1296</td>
<td>863</td>
<td>17</td>
<td>15</td>
<td>2802</td>
<td>11478</td>
<td>117</td>
<td>33</td>
<td>29</td>
</tr>
<tr>
<td>1606</td>
<td>2757</td>
<td>447</td>
<td>25</td>
<td>16</td>
<td>344</td>
<td>30311</td>
<td>154</td>
<td>53</td>
<td>35</td>
</tr>
<tr>
<td>672</td>
<td>115</td>
<td>51</td>
<td>19</td>
<td>22</td>
<td>100</td>
<td>2857</td>
<td>88</td>
<td>483</td>
<td>20</td>
</tr>
<tr>
<td>444</td>
<td>1138</td>
<td>774</td>
<td>1</td>
<td>17</td>
<td>40</td>
<td>9049</td>
<td>6</td>
<td>430</td>
<td>1</td>
</tr>
<tr>
<td>888</td>
<td>158</td>
<td>74</td>
<td>58</td>
<td>22</td>
<td>146</td>
<td>5059</td>
<td>258</td>
<td>37</td>
<td>42</td>
</tr>
<tr>
<td>1130</td>
<td>1260</td>
<td>937</td>
<td>52</td>
<td>19</td>
<td>2781</td>
<td>23643</td>
<td>274</td>
<td>156</td>
<td>61</td>
</tr>
<tr>
<td>1074</td>
<td>556</td>
<td>330</td>
<td>35</td>
<td>26</td>
<td>134</td>
<td>38133</td>
<td>134</td>
<td>44</td>
<td>23</td>
</tr>
<tr>
<td>1319</td>
<td>5199</td>
<td>3186</td>
<td>225</td>
<td>11</td>
<td>15415</td>
<td>366510</td>
<td>2088</td>
<td>509</td>
<td>393</td>
</tr>
<tr>
<td>1037</td>
<td>496</td>
<td>462</td>
<td>534</td>
<td>31</td>
<td>624</td>
<td>2634</td>
<td>1707</td>
<td>58</td>
<td>297</td>
</tr>
<tr>
<td>1074</td>
<td>2890</td>
<td>2196</td>
<td>13</td>
<td>21</td>
<td>4697</td>
<td>28392</td>
<td>63</td>
<td>170</td>
<td>13</td>
</tr>
<tr>
<td>1090</td>
<td>999</td>
<td>887</td>
<td>14</td>
<td>1</td>
<td>749</td>
<td>4882</td>
<td>1528</td>
<td>49</td>
<td>109</td>
</tr>
<tr>
<td>1361</td>
<td>37271</td>
<td>18622</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>283478</td>
<td>2</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>1028</td>
<td>821</td>
<td>660</td>
<td>219</td>
<td>42</td>
<td>1446</td>
<td>5199</td>
<td>524</td>
<td>73</td>
<td>343</td>
</tr>
<tr>
<td>375</td>
<td>607</td>
<td>100</td>
<td>15</td>
<td>75</td>
<td>200</td>
<td>3861</td>
<td>20</td>
<td>203</td>
<td>12</td>
</tr>
<tr>
<td>1764</td>
<td>7630</td>
<td>5384</td>
<td>1</td>
<td>1</td>
<td>1895</td>
<td>92812</td>
<td>139</td>
<td>95</td>
<td>7</td>
</tr>
<tr>
<td>813</td>
<td>2368</td>
<td>341</td>
<td>0</td>
<td>0</td>
<td>466</td>
<td>89391</td>
<td>10</td>
<td>81</td>
<td>7</td>
</tr>
<tr>
<td>1061</td>
<td>81</td>
<td>180</td>
<td>317</td>
<td>31</td>
<td>553</td>
<td>1456</td>
<td>1025</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>723</td>
<td>441</td>
<td>13</td>
<td>11</td>
<td>46</td>
<td>10</td>
<td>57114</td>
<td>24</td>
<td>112</td>
<td>3</td>
</tr>
<tr>
<td>1044</td>
<td>3942</td>
<td>832</td>
<td>222</td>
<td>35</td>
<td>2184</td>
<td>11079</td>
<td>626</td>
<td>139</td>
<td>284</td>
</tr>
<tr>
<td>1502</td>
<td>30282</td>
<td>25939</td>
<td>2</td>
<td>67</td>
<td>18</td>
<td>49625</td>
<td>3</td>
<td>69</td>
<td>1</td>
</tr>
<tr>
<td>443</td>
<td>1659</td>
<td>1389</td>
<td>93</td>
<td>63</td>
<td>13038</td>
<td>21352</td>
<td>147</td>
<td>55</td>
<td>48</td>
</tr>
<tr>
<td>619</td>
<td>4097</td>
<td>848</td>
<td>160</td>
<td>50</td>
<td>1147</td>
<td>96940</td>
<td>324</td>
<td>168</td>
<td>204</td>
</tr>
<tr>
<td>1549</td>
<td>26247</td>
<td>19435</td>
<td>477</td>
<td>19</td>
<td>45110</td>
<td>362561</td>
<td>2547</td>
<td>178</td>
<td>558</td>
</tr>
<tr>
<td>707</td>
<td>4614</td>
<td>858</td>
<td>37</td>
<td>35</td>
<td>2883</td>
<td>54758</td>
<td>105</td>
<td>381</td>
<td>19</td>
</tr>
<tr>
<td>1087</td>
<td>594</td>
<td>149</td>
<td>35</td>
<td>26</td>
<td>331</td>
<td>41477</td>
<td>134</td>
<td>31</td>
<td>23</td>
</tr>
<tr>
<td>1162</td>
<td>700</td>
<td>104</td>
<td>43</td>
<td>17</td>
<td>60</td>
<td>25625</td>
<td>249</td>
<td>43</td>
<td>60</td>
</tr>
<tr>
<td>1300</td>
<td>230</td>
<td>566</td>
<td>50</td>
<td>19</td>
<td>2808</td>
<td>10196</td>
<td>259</td>
<td>37</td>
<td>58</td>
</tr>
<tr>
<td>1136</td>
<td>601</td>
<td>113</td>
<td>37</td>
<td>11</td>
<td>268</td>
<td>3653</td>
<td>337</td>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td>728</td>
<td>5562</td>
<td>513</td>
<td>3</td>
<td>75</td>
<td>735</td>
<td>716148</td>
<td>4</td>
<td>477</td>
<td>3</td>
</tr>
<tr>
<td>1042</td>
<td>4268</td>
<td>2260</td>
<td>657</td>
<td>34</td>
<td>15258</td>
<td>37895</td>
<td>1950</td>
<td>63</td>
<td>158</td>
</tr>
<tr>
<td>1134</td>
<td>634</td>
<td>3757</td>
<td>583</td>
<td>34</td>
<td>26751</td>
<td>4707</td>
<td>1715</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>1066</td>
<td>2037</td>
<td>1436</td>
<td>317</td>
<td>32</td>
<td>1995</td>
<td>194581</td>
<td>1005</td>
<td>301</td>
<td>363</td>
</tr>
<tr>
<td>425</td>
<td>395</td>
<td>105</td>
<td>31</td>
<td>48</td>
<td>46</td>
<td>8871</td>
<td>64</td>
<td>23</td>
<td>17</td>
</tr>
<tr>
<td>425</td>
<td>477</td>
<td>65</td>
<td>18</td>
<td>50</td>
<td>182</td>
<td>2937</td>
<td>36</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>388</td>
<td>458</td>
<td>57</td>
<td>10</td>
<td>56</td>
<td>320</td>
<td>1217</td>
<td>18</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>1149</td>
<td>1280</td>
<td>964</td>
<td>143</td>
<td>52</td>
<td>476</td>
<td>75014</td>
<td>273</td>
<td>170</td>
<td>91</td>
</tr>
<tr>
<td>1648</td>
<td>2762</td>
<td>1872</td>
<td>138</td>
<td>46</td>
<td>1410</td>
<td>31566</td>
<td>300</td>
<td>101</td>
<td>71</td>
</tr>
<tr>
<td>1745</td>
<td>1279</td>
<td>935</td>
<td>773</td>
<td>17</td>
<td>1163</td>
<td>299808</td>
<td>4599</td>
<td>494</td>
<td>615</td>
</tr>
<tr>
<td>1701</td>
<td>6716</td>
<td>3245</td>
<td>238</td>
<td>11</td>
<td>5429</td>
<td>598477</td>
<td>2126</td>
<td>338</td>
<td>310</td>
</tr>
<tr>
<td>584</td>
<td>5123</td>
<td>4365</td>
<td>67</td>
<td>33</td>
<td>13815</td>
<td>20537</td>
<td>205</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>952</td>
<td>4110</td>
<td>3511</td>
<td>22</td>
<td>24</td>
<td>248</td>
<td>14842</td>
<td>92</td>
<td>40</td>
<td>25</td>
</tr>
<tr>
<td>772</td>
<td>12306</td>
<td>3573</td>
<td>67</td>
<td>69</td>
<td>3147</td>
<td>104997</td>
<td>97</td>
<td>359</td>
<td>37</td>
</tr>
</tbody>
</table>
Appendix B: Abstracts of Supporting Papers

Abstract: [Context] Software developers often spend a significant portion of their resources resolving submitted evolution issue reports. Classification or prediction of issue lead time is useful for prioritizing evolution issues and supporting human resources allocation in software maintenance. However, the predictability of issue lead time is still a research gap that calls for more empirical investigation. [Aim] In this paper, we empirically assess different types of issue lead time prediction models using human factor measures collected from issue tracking systems. [Method] We conduct an empirical investigation of three active open source projects. A machine learning based classification and statistical univariate and multivariate analyses are performed. [Results] The accuracy of classification models in ten-fold cross-validation varies from 75.56% to 91%. The R2 value of linear multivariate regression models ranges from 0.29 to 0.60. Correlation analysis confirms the effectiveness of collaboration measures, such as the number of stakeholders and number of comments, in prediction models. The measures of assignee past performance are also an effective indicator of issue lead time. [Conclusions] The results indicate that the number of stakeholders and average past issue lead time are important variables in constructing prediction models of issue lead time. However, more variables should be explored to achieve better prediction performance.

Abstract: Benefiting from inter-organizational collaboration while remaining competitive are organizations’ target in software industry nowadays. However, little is known about competition awareness and collaboration processes and practices in software-intensive organizations. This paper introduces the first step to understand the role and impact of competition on team collaboration in the context of inter-organizational software project. A case study was designed and initially performed. Initial findings revealed a set of propositions for future research. We found that collaboration initiates from technical coordination requirement and is influenced by technical, social and organizational factors. Meanwhile, competition comes from business strategy and competition awareness could negatively influence effectiveness of social-technical collaboration. Future works will focus on the mutual influence between competition and collaboration and role of global dispersion on this interaction.


Abstract: Adoption of team coordination practices and management of global dispersion dimensions are crucial for successful global development projects. Although considerable research effort has been made in this area, as yet, no agreement has been reached about the impact of dispersion on team coordination and performance. This paper’s objective is to identify the relationship among dispersion, team coordination and performance in global software projects. We have performed a Systematic literature review (SLR) to collect relevant studies and a thematic analysis to synthesize the extracted data. We found 28 primary studies reporting the impact of often-mixed-together five dispersion dimensions on team performance. Our results show that choices of performance measure and level of analysis can influence the direction of the effects dispersion dimensions on performance. Temporal dispersion has in general a negative impact on the perception of team on their performance while its actual impact (when directly measured from the project archive) is positive. Studies at task and team level of analysis also show negative impact of geographical dispersion on performance. Adoption of improving organic coordination practices and appropriate mechanistic coordination strategy mitigates the negative impact of dispersions.
Appendix C: Interview Guideline in Study S2
Part 1.a: Eleven Background Questions on Company and Person
(Filled in by Respondent alone, company/personal information used internally only)

1.a.1 Current date (dd.mm.yyyy):
1.a.2 Your company/local business unit (given a acronym):
1.a.3 Company URL (only used internally):
1.a.4 Number of company employees, part-time and full-time:

Below are Person data:
1.a.5 First name and last name (given a acronym):
1.a.6 Email address (only used internally):
1.a.7 Phone number (only used internally):
1.a.8 Years of Age (only used internally):
1.a.9 Degree of highest completed education, and in what area:
1.a.10 Current job position:
1.a.11 Describe your experience with software development and with OSS (tasks, products, duration etc.)!

Part 1.b: Background Questions on Project and System
(Filled in by Respondent alone)

The study object for this survey is a software development project, with at least one release of the corresponding product, and with reuse of one or more OSS components. If you have experience with several such projects, please select the one that you are most familiar with and motivated to discuss: (It may be that your company will use this OSS-based product as part of a product line, i.e. it is used in many similar software projects.)

1.b.1 What was the mean annual staff-size of the project (both full- and part-time employees)?
1.b.2 How many staffs had previous experience with OSS-based development? [ca. %]:
1.b.3 Did you have previous experience with OSS-based development before joining the project? [Mark Yes or No]:
1.b.4 What was the total effort of the project? [person-months]:
1.b.5 What was (roughly) the starting time of the project? [mm/yyyy]:
1.b.6 What was the time of the first complete delivery from the project? [mm/yyyy]:
1.b.7 What were the major application domain(s) of the system?
[Mark one or more alternatives]

<table>
<thead>
<tr>
<th>1.b.7.a</th>
<th>Bank/ Finance/ Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.b.7.b</td>
<td>Other private services (wholesale, retail, entertainment, …)</td>
</tr>
<tr>
<td>1.b.7.c</td>
<td>Public sector (health, education, government)</td>
</tr>
<tr>
<td>1.b.7.d</td>
<td>ICT industry</td>
</tr>
<tr>
<td>1.b.7.e</td>
<td>Traditional industry/engineering/construction</td>
</tr>
<tr>
<td>1.b.7.f</td>
<td>Other (please specify): ______________________</td>
</tr>
</tbody>
</table>

1.b.8 Where did the requirements come from, cf. – Market pull vs. Technology push?
[Mark one alternative]: external customer / marketing dept. / your development team
other (please specify: __________________) / don’t know: __________________

1.b.9 How were the functional Requirements described with regard to level of detail?
[Mark one alternative]: very sketchy/ coarsely/ medium/ detailed/ very fine-grained/ don’t know: ____________
and similar wrt. notation?
Mark one or more alternatives]: free text/ structured text/ use cases/ flow/state diagrams/ test cases/ other (please specify: ______________)/ don’t know: ______________

1.b.10 What was the overall, software development process/environment of the project?
[Use suitable keywords]
Ex. Lifecycle model (ad-hoc, waterfall, incremental, agile etc.) and cost model, design language (UML?), programming languages (Java with JDK?), middleware (plug-in support?), misc. development tools (Eclipse, SVN versioning tool, Bugzilla, test tools etc.), nomadic (“OSS style”) etc.

1.b.11 Brief System description
a) What were the main functionalities of the system?

b) What was the overall system-architecture pattern/ execution-environment? Ex. monolithic program, layered architecture, peer-to-peer, data-centered architecture (blackboards with cooperating agents), client-server, 3-tier architecture, application server, pipes-and-filters etc. – and software reuse either by code libraries, Web services, Service Oriented Architectures, or plug-ins.

c) What was decided first – application platform and architecture vs. major Components?

d) Was it sometimes impossible to find relevant documentation and understand the structure of some of the major Components, cf. points 2.3 and 4.1 later?
Part 2. Identify initially some OSS Component candidates that may satisfy the Requirements

(Filled in by Respondent alone, discussed a bit later)

(Use an “X”-sign to mark answers in later tables, like in 2.2 below.)

2.1. In which lifecycle phases were such OSS Components selected? [Mark one or possibly several alternatives]: Requirements/ Analysis/ Design/ Coding/ Other (please specify:_____) / Don’t know. - And give comments: ________________________________________________________________

2.2. How was the search process and initial evaluation for such OSS Components done? [Mark one or more alternatives]

<table>
<thead>
<tr>
<th>2.2.1</th>
<th>No real search/evaluation needed, since I have successfully used this component before.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2</td>
<td>Consulted with my job colleagues/engineers, or with similar persons elsewhere.</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Consulted with the customer about company policies or preferences.</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Read/heard about it in the &quot;peer-review&quot; literature: scientific conferences, journals, books etc.</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Read/heard about it in the &quot;grey&quot; literature: reports, ads, bulletin boards etc.</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Heard about it in connection with trade fairs, seminars, workshops, courses etc.</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Searched for it in general portals (<a href="http://Sourceforge.net">http://Sourceforge.net</a>), or in domain-specific ones.</td>
</tr>
<tr>
<td>2.2.8</td>
<td>Searched for it using general or specialized search engines: google, google code etc.</td>
</tr>
<tr>
<td>2.2.9</td>
<td>Used a formal method named ________ to search for and assess candidate components.</td>
</tr>
<tr>
<td>2.2.10</td>
<td>Used some systematic/documentated method named:________________________.</td>
</tr>
<tr>
<td>2.2.11</td>
<td>In another way (please specify): ____________.</td>
</tr>
</tbody>
</table>

Please comment on this search/evaluation process, f. ex, did it differ for different components?: ________________________________________________________________

2.3. What were the main information sources in deciding whether the OSS Component candidates from point 2.2 could (partly) match your functional Requirements?

   a) Component / API / architecture documentation: base for functionality-match decision; augmented by problems related to trustworthiness, code availability (not down-loadable?), incompleteness, understandability, recent changes, insufficient help/clarification given from the Community etc.

   b) Requirements specifications: not stabilized or agreed-upon, incomplete, not prioritized, hard to understand, insufficient help/clarification from the Customer etc.

2.4. Upstart effort for Part 0, Part 1, and Part 2: ______________ [minutes].

303
Part 3: Final evaluation and decision process to resolve possible Requirements mismatches vs. OSS Components

(Filled in by Researcher contact based on interview)

3.1 What did you do when the functional Requirements could not be sufficiently matched by OSS Component candidates?
   a) Consulted with the Customer to relax/modify or postpone some requirements:
   b) As a), but without clarification from the Customer:
   c) Negotiated with the Community Provider(s) and other Integrators to modify/extend existing OSS Components/APIs:
   d) Accepted that the given OSS Components/APIs represented a de-facto standard, and thus your Requirements had to be relaxed/modified etc.:
   e) No available Components to even partially matching the Requirements were found: tried then to recruit new partners to jointly develop new Components/APIs, i.e. to create a new Community, or to investigate COTS instead.
   f) Made the needed addware or Component changes locally, i.e. outside a Community:
   g) Other alternatives or activities (please specify):

3.2 How well were the major non-functional Requirements (“quality attributes”) achieved?
   (E.g. Time-to-market (duration), Cost (mainly effort), Reliability (chance of bug-free execution), Security (chance of no intruders), Safety (chance of no damage), Performance (speed, space, throughput), Usability (e.g. GUI), Maintainability (incl. Portability), New functionality (first market launch), Improved functionality (over competitors), Other (please specify:___________) And please comment!

3.3 Focusing on the 5 most important functionalities from the Requirements, can you name and explain the matching OSS Components that you finally integrated into your system? (disregarding "standard" OSS Components, like Hibernate, Spring etc.):
   Component 1: __________________________________________
   Component 2: __________________________________________
   Component 3: __________________________________________
   Component 4: __________________________________________
   Component 5: __________________________________________
   How many such OSS Components have been integrated so far?__________
   How would you say the final Component choices matched the requested functionalities?

3.4 How big part of the system do the OSS Components now occupy? [roughly in %, and counted in #LOC or #features]: ________________________________
Part 4: All-over comments on the cooperative work in the most-used OSS Communities

(Filled in by Researcher contact based on interview)

4.1. What was the cooperative work process when integrating the chosen OSS Components into your Application code?
   a) What communication tools were used, f.ex., phone, Skype, email (lists), IRC, bulletin boards?
   b) Was there a dedicated contact person in your team to follow up each relevant Component/Community?
   c) technical incompatibilities wrt. software architecture, control flow/ tasking, exception handling, data formats etc. – And was extra glueware needed, or was technical/architectural documentation so incomplete, that this Component had to be dropped?
   d) Did you need and seek help from the Community to clarify APIs etc.?
   e) What was the process when you announced a bug report? – A quick response?________
   f) How to locate and fix bugs in own code vs. addware/glueware vs. OSS Components?
   g) How to estimate integration/ testing costs?
   h) Other issues (please specify)? ______________________

4.2. Did changed OSS Components or functional Requirements create any problems?
   a) What was a typical negotiation process and outcome after you announced a change request for a shared Component/API, due to changed business needs and ensuing Requirements on your side?
   b) What was the corresponding negotiation process when other Integrators or even Providers announced their change requests, that might affect you?
   c) How stable were the OSS Components you mostly used – over time?
   d) Other issues (please specify)?

4.3. What were your team’s main contributions to and feedbacks from the Community?
   a) Own bug reports:
   b) Bug fixes with code to repair a bug:
   c) Own change requests for modified or new Components/APIs:
   d) Working code for such change requests:
   e) You became a co-Provider (actively working in an existing Community):
   f) You became a main Provider (starting a completely new Community):
   g) Changed some of the de-facto, cooperative work processes in a Community:
   h) Participated in national OSS organizations, f.ex. to promote an “OSS culture”:
   i) Other issues (please specify)?:

4.4. Did licensing terms turn out satisfactorily, or were special negotiations needed towards the Customer/Community?
Part 5: General sum-up and reflections

(Filled in by Researcher contact based on interview)

5.1. What were the main motivations for using OSS in this project?

E.g. General reasons: strategic importance, being “political correct”, learning/training/experience wrt. “hot” technologies, risk assessment, getting external PR, easier to recruit new staff, explicit decision by the Customer, implicit decision by the developers etc.

E.g. Concrete benefits: cost cuts through software reuse, avoiding “lock-in” from proprietary 3-party vendors, simplified procurement and Community support, immediate code availability, often a more modern architecture etc.

E.g. Concrete Costs: extra integration and testing work (“glueware”), lack of matching OSS products – so need to customize (Requirements vs. Components!), mistrust of Provider/Community’s expertise and support, impact of licensing constraints etc.

5.2. Which factors influenced your prioritization of Requirements vs. OSS Components?

Possible factors: clear business strategies, established guidelines for de-facto OSS-standardization, architectural flexibility, provider reputation, well-functioning Communities with friendly work atmospheres, initial coverage of core functionalities, dedicated follow-up of non-functional requirements etc.

5.3. Final assessment of cooperative work?

- In general OK or not? Was there a balanced “give-and-take” – in the short and long run?
- What were the key success criteria, cf. all above issues?
- Did attitudes to OSS change over time? – how and why?
- Will you recommend your company to engage in further OSS-based projects? …

5.4 Any interesting side effect (spin-offs?)

- Did you engage in other OSS communities over time – with new software partners?
- Did you by your OSS work engage in other areas over time – new businesses or products?
- Other unexpected effects that turned interesting? …

5.5 Other comments?
Appendix D: Conceptual Scheme in Study S3
Appendix E: Interview Guideline in Study S4
Questions on current practices on Team collaboration

Who do you communicate with in the team?

- **R1:** Team member (same or different site) [ ] (Yes or No)
- **R2:** Project manager [ ] (Yes or No)
- **R3:** Customer [ ] (Yes or No)
- **R4:** Open source community [ ] (Yes or No)

Detail on team collaboration practices

(*If R1 is yes*) Collaboration with team member

1. Do you know the tasks of other team members? [ ] (Yes or No)
2. What are the methods used to communicate within your team?
   - **T1:** Online collaboration support tool [ ] (Yes or No)
   - **T2:** Email [ ] (Yes or No)
   - **T3:** Issue tracking system [ ] (Yes or No)
   - **T4:** Telephone meeting [ ] (Yes or No)
   - **T5:** Face to face meeting & site visits [ ] (Yes or No)
   - **T6:** Discussion board [ ] (Yes or No)
3. What kinds of information are exchanged?
   - Information about task [ ] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about team member [ ] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about process [ ] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about formality [ ] (Yes or No. If yes: T1/ T2…/ T6)
   - RE/ design/ maintenance [ ] (Yes or No. If yes: T1/ T2…/ T6)
4. How frequent are the communication? (number of emails/ per day, number of discussion per day/ number of phone call per day …)
5. How the task is divided in the team?
6. How the task is discussed and decided in the team?

(*If R2 is yes*) Collaboration with Project manager

7. What is the role of project manager in team?
8. What are the methods used to communicate within your team?
   - **T1:** Online collaboration support tool [ ] (Yes or No)
   - **T2:** Email [ ] (Yes or No)
   - **T3:** Issue tracking system [ ] (Yes or No)
   - **T4:** Telephone meeting [ ] (Yes or No)
   - **T5:** Face to face meeting & site visits [ ] (Yes or No)
   - **T6:** Discussion board [ ] (Yes or No)
9. What kinds of information are exchanged?
   - Information about task [ ] (Yes or No. If yes: T1/ T2…/ T6)
- Information about team member [___] (Yes or No. If yes: T1/ T2…/ T6)
- Information about process [___] (Yes or No. If yes: T1/ T2…/ T6)
- Information about formality [___] (Yes or No. If yes: T1/ T2…/ T6)
- RE/ design/ implementation/ maintenance [___] (Yes or No. If yes: T1/ T2…/ T6)

10. How frequent are the communication? (meetings per week, months, …)

(If R3 is yes) Collaboration with customer

11. What is the role of project manager in team?
12. What are the methods used to communicate within your team?
   - T1: Online collaboration support tool [___] (Yes or No)
   - T2: Email [___] (Yes or No)
   - T3: Issue tracking system [___] (Yes or No)
   - T4: Telephone meeting [___] (Yes or No)
   - T5: Face to face meeting & site visits [___] (Yes or No)
   - T6: Discussion board [___] (Yes or No)

13. What kinds of information are exchanged?
   - Information about task [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about team member [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about process [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about formality [___] (Yes or No. If yes: T1/ T2…/ T6)
   - RE/ design/ implementation/ maintenance [___] (Yes or No. If yes: T1/ T2…/ T6)

14. How frequent are the communication? (number of emails/ per day, number of discussion per day/ number of phone call per day …)

(If R4 is yes) Collaboration with OSS community

15. What is the role of OSS community in your project?
16. What are the methods used to communicate within your team?
   - T1: Online collaboration support tool [___] (Yes or No)
   - T2: Email [___] (Yes or No)
   - T3: Issue tracking system [___] (Yes or No)
   - T4: Telephone meeting [___] (Yes or No)
   - T5: Face to face meeting & site visits [___] (Yes or No)
   - T6: Discussion board [___] (Yes or No)

17. What kinds of information are exchanged?
   - Information about task [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about team member [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about process [___] (Yes or No. If yes: T1/ T2…/ T6)
   - Information about formality [___] (Yes or No. If yes: T1/ T2…/ T6)
   - RE/ design/ implementation/ maintenance [___] (Yes or No. If yes: T1/ T2…/ T6)
Appendix F: Data Extraction Scripts in Study S5
use strict;
use Switch;
use Data::Dumper;
open B, "list_of_repo.txt";
our @repolist = ();
while(<B>)
{
    chomp();
    push(@repolist,$_);
}
my %tot_no_of_files = ();
my %no_of_copied_files = ();
my %no_of_copied_files_before = ();
my %no_of_copied_files_after = ();
my %no_of_copied_files_parallel = ();
my %no_of_unique_files = ();
my %count3 = ();
my %lastdate = ();
my %firstdate = ();
my %no_of_related_files = ();
my %countnoofcommit = ();
my %countnoofmr = ();
my %branchlist = ();
my %no_redundant_mr = ();
foreach my $line (@repolist)
{
    $tot_no_of_files{$line}=0;
    $no_of_copied_files{$line}=0;
    $no_of_copied_files_before{$line}=0;
    $no_of_copied_files_after{$line}=0;
    $no_of_copied_files_parallel{$line}=0;
    $count3{$line}='';
    $branchlist{$line} = "";
    $lastdate{$line} = 0;
    $firstdate{$line} = 20151212;
    $countnoofmr{$line} = 0;
    $no_of_related_files{$line}=0;
    $countnoofcommit{$line}=0;
    $no_of_unique_files{$line}=0;
    $no_redundant_mr{$line}=0;
}
open A, "all.summ2";
print "Done 1 ! \n";
my $count =0;
open C, ">>detail_repo_15062014.txt";
open D, ">>detail_svvm.txt";
while(<A>)
{
    chomp();
    my ($filename, $noofvers, $curloc, $noofauthors, $noofauthorsleft, $noofcfds,
    $noofmrs, $unknown1, $maxloc, $isosss, $relatedcfds, $relatedmrs, $relatedauthors,
    $latestmodifieddr, $latestrelatedmr, $unknown2, $latestcommiter,
    $latestrelatedmodifiedfile, $scfdtype, $unknown3, $latestrelatedmr, $unknown4,
    @listofcommitters) = split(/\n,\$,\-1);
#my @paths = split("/", $filename, -1);
#my $repo = join("/", $paths[0..1]);
#my $repo = $paths[0];
$count++;
for(my $i = 0; $i < scalar(@repolist); $i++){
  my $repo = $repolist[$i];
  my $no_of_copied_files_flag=0;
  my $no_of_copied_files_before_flag=0;
  my $no_of_copied_files_after_flag=0;
  my $no_of_copied_files_parallel_flag=0;
  my $tmp = index($filename, $repo);
  #print "$filename-$repo\n";
  if(index($filename,$repo) eq 0)
  {
    #get the rest of the string after $filename
    my $repospath = "";
    #print "y ";
    my $tempdate = parse_datetime($latestmodifieddate);
    if($tempdate ge $lastdate{$repo}){
      $lastdate{$repo} = $tempdate;
    }
    if($tempdate le $firstdate{$repo}){
      $firstdate{$repo} = $tempdate
    }
    # count number of file in the repo
    $tot_no_of_files{$repo}++;;
    my $stmp = $countnoofmr{$repo};
    $countnoofmr{$repo} = $stmp + $noofmrs+0;
    # loop through the list of related files:
    if(scalar (@listofcommiters) == 0){
      $no_of_unique_files{$repo}++;
    }
    my %tempcount3 = ();
    foreach my $line (@listofcommiters){
      my @fileinfo = split('=', $line);
      my @relatedpath = split('/', $fileinfo[0]);
      my @currentfileinfo = split('=', $listofcommiters[0]);
      # with a related file from different repos
      if(@relatedpath[0] ne $repo)
      {
        # for a related file from different repositories, count a
        # link between two repositories, count a link if two repositories are in parallel development,
        # count committers in common between them
        $no_of_copied_files_flag=1;
        ($no_of_copied_files_before_flag,
        $no_of_copied_files_parallel_flag, $no_of_copied_files_after_flag) = createinterrepoRelation(@relatedpath[0], @fileinfo[1],@fileinfo[2],$repo, @currentfileinfo[1],
        @currentfileinfo[2]);
        if(defined $tempcount3{$repo}){
          addtogroup($tempcount3{$repo},$relatedpath[0]);
        }
      }
    }
{  
    $tempcount3{$repo} = $relatedpath[0];
}
addtogroup($count3{$repo}, $relatedpath[0]);
#count number of commits to the related file
$countnoofcommit{$repo} += $fileinfo[3];
}
else{
    # count number of related files in all (within the repository and
to others)
    $no_of_related_files{$repo}++;
}
my $tmp1 = $no_redundant_mr{$repo};
my $tmp3 = scalar( split( ' ' , $tempcount3{$repo} ) ) + 0;
my $tmp2 = 0;
if($tmp3 gt 0){
    $tmp2 = ($noofmrs * ($tmp3 - 1)) / $tmp3;
} #print "$tmp2: ";
$no_redundant_mr{$repo} = $tmp1 + $tmp2;
if($no_of_copied_files_flag == 1){
    $no_of_copied_files{$repo}++;
}
if($no_of_copied_files_before_flag == 1){
    $no_of_copied_files_before{$repo}++;
}
if($no_of_copied_files_after_flag == 1){
    $no_of_copied_files_after{$repo}++;
}
if($no_of_copied_files_parallel_flag == 1){
    $no_of_copied_files_parallel{$repo}++;
    if(index($filename, "svvm") eq 0){
        print D "$filename \n";
    }
}
}
print C "Project; First change; Latest change; No Mr per repo; Number of files; Number of MRs; Number of cross-projected related files; Number of inner project related files; Number of commits in cross-project related files; Related projects; Number of unique files; Number of files changed before; Number of files changed after; Number of files changed in parallel \n";
foreach my $line2 (@repolist){
    print C "$line2; $firstdate{$line2}; $lastdate{$line2}; $no_redundant_mr{$line2}; $tot_no_of_files{$line2}; $countnoofmr{$line2}; $no_of_copied_files{$line2}; $no_of_related_files{$line2}; $countnoofcommit{$line2}; $count3{$line2};";
}
sub addtogroup
{
  my $flag=1;
  my @repos = split(' ',$_[0]);
  foreach my $repo (@repos)
  {
    if($_[1] eq $repo)
    {
      $flag = 0;
    }
  }
  if($flag == 1)
  {
    $_[0]=join(' ',$_[0], $[1]);
  }
}

sub createinterreporelation
{
  my $my_no_of_copied_files_before_flag = 0;
  my $my_no_of_copied_files_parallel_flag = 0;
  my $my_no_of_copied_files_after_flag = 0;
  my $flag = checkdurationoverlap($_[1],$_[2],$_[4],$_[5]);
  switch ($flag) {
    case 1 {
      $my_no_of_copied_files_parallel_flag = 1;
    }
    case 0 {
      $my_no_of_copied_files_after_flag = 1;
    }
    case 4 {
      $my_no_of_copied_files_before_flag = 1;
    }
    case 2 {
      $my_no_of_copied_files_parallel_flag = 1;
    }
    else {
    }
  }
  return ($my_no_of_copied_files_before_flag, $my_no_of_copied_files_parallel_flag,
  $my_no_of_copied_files_after_flag);
}

sub checkdurationoverlap
my $start1 = parse_datetime($_[0]);
my $end1 = parse_datetime($_[1]);
my $start2 = parse_datetime($_[2]);
my $end2 = parse_datetime($_[3]);
if($start1 gt $end2)
{
    # 1 after 2
    return 0;
}
elseci (($start1 ge $start2) && ($start1 le $end2) && ($end2 ge $start1) && ($end2 le $end1))
{
    return 1;
}
elseci (($start2 ge $start1) && ($start2 le $end1) && ($end1 ge $start2) && ($end1 le $end2))
{
    return 1;
}
elseci ($start2 gt $end1)
{
    return 4;
}
return 2;

sub parse_datetime
{
    my ($Y,$m,$d) =split('-', $_[0]);
    return "$Y$m$d";
}

sub convert_datetime
{
    return join("-", substr($_, 0, 4), substr($_, 4, 2), substr($_, 6, 2));
}
Appendix G: SNA of Spark Projects in Study S5