Noname manuscript No.
(will be inserted by the editor)

Handover of Managerial Responsibilities in Global Software Development: A Case Study of Source Code Evolution and Quality

Ronald Jabangwe · Jürgen Börstler · Kai Petersen

Received: date / Accepted: date

Abstract *Context:* Studies report on the negative effect on quality in global software development (GSD) due to communication and coordination related challenges. However, empirical studies reporting on the magnitude of the effect are scarce. *Research Method:* This paper presents findings from an embedded explanatory case study on the change in quality over time, across multiple releases, for products that were developed in a GSD setting. The GSD setting involved periods of distributed development between geographically dispersed sites as well as a handover of project management responsibilities between the involved sites. Investigations were performed on two medium-sized products from a company that is part of a large multinational corporation. Quality is investigated quantitatively using defect data, and measures that quantify two source code properties, size and complexity. Observations were triangulated with subjective views from company representatives. *Results and Findings:* There were no observable indications that the distribution of work or handover of project management responsibilities had an impact on quality on both products. Among the product-, process-, and people-related success factors, we identified well-designed product architectures, early handover planning, support from the sending site to the receiving site after the handover, and skilled employees at the involved sites. Overall, these results can be useful input for decision-makers who are considering distributing development work between globally dispersed sites or handing over project management responsibilities from one site to another. Moreover, our study shows that analyzing the evolution of size and complexity properties of a product's source code can provide valuable information to support decision-making during similar projects. Finally, the strategy used by the company to relocate responsibilities

Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Tel.: +46 455 38 50 00
Fax: +46 455 38 50 57

Ronald Jabangwe E-mail: ronald.jabangwe@bth.se ·
Jürgen Börstler E-mail: jurgen.borstler@bth.se ·
Kai Petersen E-mail: kai.petersen@bth.se

can also be considered as an alternative for software transfers, which have been linked with a decline in efficiency, productivity and quality.

Keywords Global Software Development · Distributed Development · Source Code Analysis · Software Transfers · Object-oriented Measures · Case Study

1 Introduction

Globalization has led to an increase of software that is developed and evolved in global software development settings (GSD) (Aspray et al., 2006; Šmite et al., 2010). GSD is characterized by highly geographically dispersed teams working on developing the same software product. GSD settings have been reported to negatively impact quality, for example due to challenges in sustaining effective levels of communication and coordination practices (Ramasubbu and Balan, 2007). Thus attaining the potential GSD benefits, such as cost reduction, should not be taken for granted (Conchúir et al., 2006). Findings from empirical investigations on the implications of GSD settings on quality would be valuable input in the decision-making process for practitioners and help them with planning and managing future projects. However, empirical studies reporting on the implications on quality are scarce.

In this study we investigate two products that were developed in a GSD setting that involved distributed development teams. Furthermore, after the setting of distributed software development, project management responsibilities for the two products were also gradually shifted from one site to another. We refer to this period with the shift or move as the handover period. This handover period can be compared to the transfer period that is observed in software transfers. A software transfer is the relocation of development and maintenance work from one site to another (Šmite and Wohlin, 2011; Wohlin and Šmite, 2012). The handover activities in our study do not constitute a transfer by definition. Nevertheless, literature reports that relocation of work that occurs during software transfers can negatively impact quality (Mockus and Weiss, 2001; Šmite and Wohlin, 2012; Jabangwe and Šmite, 2012). Thus, in our case, we investigate the impact on quality of both the distributed settings and the handover activities between geographically dispersed sites.

An explanatory case study approach, as described by Runeson et al. (2012), is used to provide insights in this study. Quality is evaluated using measures extracted from the source code and defect reports that are linked to problems in the source code. Source code measures are used because they are reported to be good predictors of quality (Basili et al., 1996; Bansiya and Davis, 2002; Jabangwe et al., 2013). To arrive at a more general conclusion two different commercial products from the case company are studied.

To understand the impact of distributed development and handover of responsibilities on the quality of source code, we investigate the evolution of source code measures across releases, before, during and after a handover. In addition, this investigation is done using a longitudinal perspective by capturing and highlighting major events that occurred during the evolution of the two products. This extensive analysis is performed to understand the extent of the impact on quality of distributed development and the handover activities in relation to other events.

Based on our study of the literature (Spinellis, 2006; Ramasubbu and Balan, 2007; Bird et al., 2009; Verner et al., 2012), an investigation on this magnitude in a GSD context differs from other studies because it makes the following contributions:

- The study investigates software quality during evolution and its potential relation to distributed development and handover of project management responsibilities. To examine variations in internal product quality, measures are used that quantify the size and complexity of the source code, from multiple releases. These measures are referred to as source code measures in our this study. Source code measures are measures that are obtainable from source code (e.g., number of classes, which quantifies the size property of source code).
- Study of relationships between source code measures and external software quality (measured as externally reported defects), the time between releases, and the number of new functionality per release, in a GSD context.

The remainder of this paper is organized as follows. Related work is outlined in Section 2. Section 3 describes the case company, and research questions are presented in Section 4. Data collection and data analysis approaches are discussed in Section 5. Our findings are presented in Section 6 and discussed in Section 7. Threats to validity of our study are discussed in Section 8. Lastly, conclusions and future work are presented in Section 9.

2 Related Work

There are studies that report on the effect of organizational structure on quality. For example, Nagappan et al. (2008) found that there is a strong link between the software development organizational structure and code quality. In GSD contexts, organizational structures are often characterized by distributed or dispersed teams, which can make work challenging. Communication and coordination, elements that are important for an effective organizational structure (Brooks, 1995), are difficult to maintain efficiently in a GSD setting (Carmel, 1999). The next subsection provides an overview of studies that have reported implications on quality of GSD settings.

2.1 GSD and Quality

Ramasubbu and Balan (2007) found that distributed development had a negative impact on quality in their studied case. Customer problem reports were used as an indicator for conformance quality, i.e., adherence to customer specifications, in the study. Challenges that were faced in the distributed development setting hindered effective coordination practices between the sites. Because of the correlation between performance and quality, Ramasubbu and Balan found that dispersion of sites had an indirect effect on quality. However, Spinellis (2006) and Bird et al. (2009) report contradicting results.

Spinellis (2006) found that geographical dispersion of developers had a negligible effect on productivity and quality in their studied case. Spinellis used adherence, or lack there of, of coding style, and defect density as surrogates for quality. Results of the study show that developing software in a GSD context with geographically dispersed developers does not necessarily result in higher defect density in the source code. In the study, however, Spinellis used an open source system, FreeBSD, which, by its inherent characteristics, involves dispersed developers. Some of the inherent characteristics of open source systems could also play an important role in the results of their study. Unlike in GSD contexts, as Spinellis points out, developers of open

source systems are often volunteers and thus human aspects such as motivation may not be a critical factor, as it would be the case for employees in a traditional office setting.

Bird et al. (2009) found negligible differences on the effect on quality between a collocated setting and a distributed development setting. Studying Windows Vista, they found that code written by distributed developers and that written by collocated developers had about the same number of post-release failures. They also investigated the differences between binaries developed by distributed developers and those developed by collocated developers. For this, they collected the following measures from each of the binaries of Windows Vista: size (e.g., number of classes), complexity (e.g., cyclomatic complexity), code churn (e.g., lines added), test coverage and dependencies between binaries. They found that there was no difference in the measures for binaries developed in a distributed context and those developed by collocated developers. Bird et al. accredit the negligible difference in quality between the collocated and distributed development settings to practices that mitigated communication and coordination issues. Some of these practices are long-term relationships and cultural awareness between sites, and consistent use of the same tools across sites, for example synchronous communication tools and configuration tools. It is also important to note that most of the binaries were distributed within close proximity (building, campus and city).

In our study, we provide a much deeper understanding of the impact of distributed development on quality by using multiple sequential releases in the investigation. Even though Spinellis (2006) and Bird et al. (2009) suggest that distributed development may not impact quality, Nagappan et al. (2008) highlight the influence that organizational structures have on quality. There were also no changes in organization or development sites for the products studied by Spinellis (2006) and Bird et al. (2009). In the cases presented in this paper, though work is distributed during the evolution of the products, there were noticeable changes in organizational structure. More specifically, there was a handover period of project management responsibilities from one site to another. This handover period can be compared to the transfer period that occurs during software transfers. Software transfers have been linked with a decline in quality. Studies on software transfers as well as an explanation of the difference between a transfer period and the handover period in our case are discussed in the next subsection.

2.2 Impact of Transferring Activities

Though there has been an increase in empirical GSD studies (Šmite et al., 2010; Nurdiani et al., 2011; Verner et al., 2012), there are few studies that focus on critical software transfer factors or the impact of software transfers on software quality.

Mockus and Weiss (2001) report that transfers may be linked to a decrease in productivity. According to their findings, this decrease can be attributed to learning curves of a complex product. Through their case study they also identify certain transfer strategies or approaches implemented at the case company, i.e., transferring by development stage, by maintenance, by functionality or by localization. Mockus and Weiss also report that the following factors can contribute to a successful transfer: minimizing the need for communication and coordination between the sending and the receiving site by, for example, transferring decoupled software development

tasks; training the receiving site; and increasing the receiving site's product familiarity.

Using results from two transfer projects, Šmite and Wohlin (2012) highlight factors that make it difficult to sustain acceptable levels of efficiency in GSD settings that involve transfers. They categorize the factors as either product-, process- and people-related. Examples of some of the factors are cultural differences and differences in work processes between the receiving site and the sending site, and product complexity. The authors also provide recommendations for alleviating some of the issues faced during software transfers. Examples of some of the recommendations are: cultural awareness coaching, ensuring that processes fit the needs of those involved in the project, training the receiving site, and ensuring that support is available for the receiving site after the transfer. The authors also suggest that some products are better candidates for transfers than others, depending on the product's complexity and maturity, completeness of the product's documentation, and the competency of the receiving site in relation to the product. These and other characteristics are elaborated in Šmite and Wohlin (2011), using findings from four transfer projects that were conducted at Ericsson. They characterized product-, process- and people-related factors that are favorable and unfavorable factors before, during and after a transfer and produced a risk identification checklist to help with decision-making when planning and executing software transfers.

In a previous study, Jabangwe and Šmite (2012) investigated the impact of a transfer on quality across subsequent releases of a single product. To understand quality, the authors use subjective views from those involved with product development and maintenance, and objective (quantitative) defect data reported across multiple releases. Their study highlights an initial decline in quality, i.e., increase in defect data inflow, after a transfer. Subjective views on quality were also found to be consistent with observations based on the quantitative data about the product's change in quality over time.

The work that is transferred in the aforementioned studies is much more challenging than the work in the present study. In the present study, only project management responsibilities are relocated; whereas the work relocated in the studies by Jabangwe and Šmite (2012), Mockus and Weiss (2001) and Šmite and Wohlin (2012) is development and/or maintenance work. By the definition of software transfers provided by Wohlin and Šmite (2012), the handover of responsibilities in our study cannot be classified as a transfer. However, the relocation of work done in transfers is linked with negative effect on quality, and thus we conjecture that the relocation of project management responsibilities during the handover period in the present study can have similar consequences. This conjecture is based on the negative impact on quality that product learning curves and lack of competency may have during handovers as observed in software transfers. However, the context is different given that management responsibility handover was studied. Hence, another proposition could be that there is no difference as the problems of learning and understanding the transferred system are not of such a major concern. Thus, this study tests which of the two propositions holds true for the studied context.

2.3 Summary

Overall, the present study differs from the related studies on the following three points: First, the main difference is that we take a longitudinal view of changes in quality that involves development setting in distributed development, as well as handover activities from one site to another. Thus, with regards to distribution of work, we believe our study adds to the body of knowledge on the effect of distributed development on quality by providing results from a different industrial context. This information would help when conducting a meta-analysis of empirical studies to understand factors that influence quality in distributed settings, e.g., human factors or shift from one development setting to another. Given the challenging work associated with software transfers in comparison to handover of responsibilities it is important to differentiate the two. Second, we focus on multiple views on quality (in particular internal source code quality and external product quality - the studies presented in related work only focused on external quality). Third, we also considered the shift of management responsibilities and investigated whether they have an effect on the source code quality.

The previous paragraph made it more explicit how this study differs from related studies. It is important to differentiate and reveal similarities with other studies on a topic, because this helps with understanding if and how a study is contributing to the existing body of knowledge (Wieringa, 2013). Making the difference between cases more explicit also makes it easier to compare one case to another and to further understand the extent of generalizability (Wieringa, 2013; Runeson et al., 2012).

In the next section, we present the study context and the data collection and analysis methods used in this study.

3 Case Description and Context

3.1 Company Description

The case company is part of a large multinational corporation, with decades of experience providing reliable and accurate radar gauging solutions. The company provides solutions for a wide range of industries including chemical, marine, oil and gas. It has been involved in numerous globally distributed development work and is therefore interested in understanding the implications on source code quality.

Following the case study guidelines proposed by Runeson et al. (2012), we consider the present study as an embedded explanatory case study with two units of analysis from two separate products from the same company. The unit of analysis is the source code. The study is more explanatory in nature because we investigate the nature of relationships and explanation of the relationships between internal and external product measures. For anonymity reasons, the two products are referred to as Product-A and Product-B and the two sites as Site-Alpha and Site-Beta.

Brief descriptions and contextual information for the two products are provided in the following subsections.

3.2 Product-A

Product-A is a standalone, Windows-based configuration software package, with a human-machine interface, that helps with configuring tank systems that are used for, for example, liquid storage. Product-A consists of source code components developed using the programming language C++, and source code components developed using the programming language Visual Basic (VB). Figure 1 provides an overview of the timeline for Product-A.

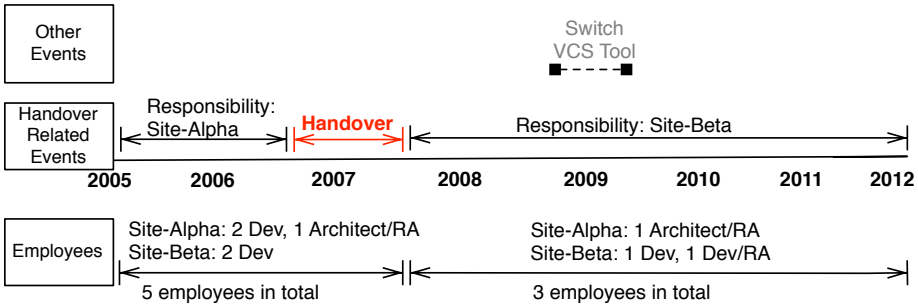


Fig. 1: Timeline for Product-A

Handover Related Activities: The product was initially developed in collaboration with an external company and then relocated to the case company. Project management responsibilities were at Site-Alpha in Sweden, between 2005 and 2006. During this time period, Site-Alpha had two developers and one architect dedicated to product development. Site-Beta had two developers contributing to product development activities.

Project management responsibilities for Product-A were gradually relocated to a site in Russia, Site-Beta, between the last quarter of 2006 and mid 2007. During this period the number of employees at each site remained the same.

After the relocation or the handover, Site-Alpha had one employee involved in architecture design and requirements analysis. Site-Beta had two developers. One of the developers at Site-Beta was also involved with requirements analysis.

Other Major Events: Between the last quarter of 2008 and mid 2009, the version control system (VCS) for Product-A was switched from the tool Visual Enabler to Subversion.

3.3 Product-B

Product-B is a Windows-based software package, with a human-machine interface, configuration and inventory management system for tank gauging systems, i.e., liquid storage tanks. By quickly and accurately, in real-time, collecting and processing tank measurement data, and providing computations such as volume and density, and features such as alarm handling, Product-B, provides a means for monitoring multiple

tanks. Most of the source code components, approximately 98%, are developed using the programming language C++. Figure 2 provides an overview of the timeline for Product-B.

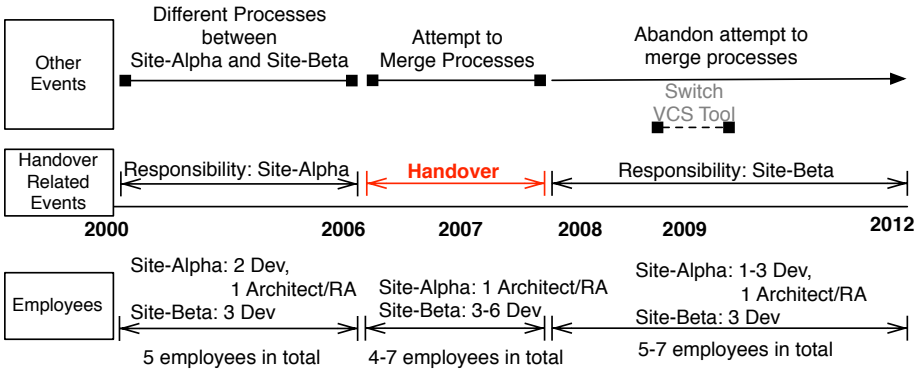


Fig. 2: Timeline for Product-B

Handover Related Activities: Project management responsibilities were at Site-Alpha, which is located in Sweden, between 2000 and 2006. Employees from Site-Beta, located in Russia, were involved in software development activities. During this time period Site-Alpha had two software developers and one other employee involved in architectural design and requirements analysis. Site-Beta had three software developers contributing to product development activities.

Project management responsibilities were gradually relocated from Site-Alpha to Site-Beta, between the first quarter of 2006 and the last quarter of 2007. During this time period, Site-Alpha had one employee involved in architectural design and requirements analysis. The number of software developers at Site-Beta gradually increased from three to six. The main reason for this increase was to speed up the development process to meet and satisfy growing customer needs.

By the beginning of 2008, project management responsibilities were at Site-Beta. Site-Beta had three software developers contributing to product development activities. Involvement of Site-Alpha was mainly on requirements analysis and architectural design activities that were assigned to one employee. However, depending on the workload, the number of developers from Site-Alpha that would get involved in product development activities varied between one and three.

Other Major Events: Before the initiation of the handover of responsibilities, Site-Alpha and Site-Beta, used different development processes. Representatives from the case company stated that during the handover, an attempt was made to merge the processes. However, due to a global economic crisis that occurred during and after the handover activities, the merging initiative was not highly prioritized and it was abandoned soon after the handover. As a result, the two sites returned to using different processes. Between the last quarter of 2008 and mid 2009, the VCS tool for both Product-A and Product-B was switched from the tool Visual Enabler to Subversion.

Since we examine product quality taking into consideration the evolution of the product, it was important to consider other major events that occurred during the evolution of both products, Product-A and Product-B. Such an inclusive approach of capturing major events helps to identify confounding factors other than the handover activities that may have affected product quality. It also helps understand the magnitude of the effect of various events, and compare the effect between events.

The following subsection summarizes the contextual information for Product-A and Product-B and discusses how the two cases relate to GSD.

3.4 Context

Context is important when assessing the generalizability of a study (Petersen and Wohlin, 2009). The following are noteworthy characteristics of the context that are evident in the two cases of the present study:

- In both cases, the sending site is located in Sweden and the receiving site is located in Russia.
- In both cases, the receiving site was involved in product development before the handover of project management responsibilities.
- In both cases, the sending site retained some front-end activities, e.g., architectural design and requirements analysis, after the handover.

Furthermore, in both cases, the reason for the handover of responsibilities was to reduce development costs. The receiving site, in both cases, was selected for its close proximity to Sweden, and low cultural differences between Sweden and Russia in comparison to alternatives in Asia. The company had done an internal analysis and discovered that the cultural distance with Russia was less than they had experienced with the site(s) in Asia. The site in Russia was also selected due to low development costs and availability of additional skilled labor in comparison to the site in Sweden.

The following are the main differences between Product-A and Product-B. Development work for Product-B was distributed before, during and after the handover, whilst for Product-A it was only distributed before and during the handover. After the handover period, Product-A's development work was collocated after the handover period (at Site-Beta).

Table 1 shows how the contexts of the two cases fit into the GSD field according to GSD scenarios, characteristics, and classifications described by Šmite et al. (2008) and Šmite and Wohlin (2011).

4 Research Questions

The present study answers the following research questions:

RQ1: *Which change patterns in internal product quality (source code measures) and external quality attributes can be identified over the case study period?*

RQ2: *In which ways do internal product quality (source code measures) and external product attributes relate to each other?*

RQ3: *Which change patterns and relationships can be attributed to particular events or incidents during product evolution?*

Table 1: Study Context

	Empirical Background
Study Perspective	Embedded case study
Sub Methods	Quantitative analysis and questionnaires
Unit of Analysis	Source code
Focus of Study	Source code quality during evolution in a GSD context, that involves distributed development and a handover of project management responsibilities between geographically separated sites
Empirical Focus	Empirically-based (explanatory)
Subject(s)	Practitioners
	GSD Background
Collaboration Mode	Intra-organization/Offshore insourcing
Approach	**Product-A:** Single-site project management responsibilities before and after a handover of the responsibilities, during product evolution. Prior to the handover, development work was distributed between the sending and receiving site. After the handover, all development activities were at one site (the receiving site). Architectural design and requirements analysis activities were shared between the sending and receiving site. **Product-B:** Single-site project management responsibilities before and after a handover, during product evolution. Before, during and after the handover, development work was distributed between the sending and the receiving site. Architectural design and requirements analysis activities were at the sending site.

RQ4: *How do the observations identified in RQ1/RQ2/RQ3 relate to the distribution of work between geographically dispersed sites?*

RQ5: *How do the observations identified in RQ1/RQ2/RQ3 relate to the handover periods, i.e., before, during and after the handover of responsibilities?*

Quantitative and qualitative data are used to answer the research questions. How the data was analyzed, is discussed in the next section.

5 Data Collection and Analysis

Releases and Source Code Components: Data was collected from releases that had at least one new functionality. We define new functionality as any capability or function added to a release, which meets certain customer needs (ISO/IEC/IEEE-24765, 2010), that was not present in the preceding releases. Thus, releases that only contained defect fixes and were considered as correctional packages were excluded from the analysis. The assumption is that releases with only correctional packages are eventually included in the next iteration of the major releases. Hence, they are indirectly examined with each major release. In addition, source code components that

did not capture the evolution of the product, such as library files, and did not contain the main functionality of the system were excluded. This exclusion was done to ensure that analysis was done only on releases and source code components that captured product evolution, i.e., the products' major releases and components. General availability dates for releases were also collected and used to construct evolutionary timelines of the products.

For Product-A, the developer in Site-Beta, who was also involved with architectural design and requirements analysis, helped with identifying major releases and core source code components. For Product-B, an architect at Product-A's Site-Alpha helped with identifying major releases and core source code components.

Source Code Static Analysis Tool: The freeware tool Source Code Monitor[1] was used to extract source code measures. It was selected because it has been available for free download and receiving enhancements for over a decade. Furthermore, the tool can quantify size and complexity properties for both C++ and VB source code. During evolution, measures of these properties are most likely to increase (Lehman, 1980) and this can have an impact on quality as the software can become more fault-prone and more difficult to maintain (Singh et al., 2009; Kanellopoulos et al., 2010). Furthermore, measures of size and complexity show more consistent links with external quality attributes than measures of other properties, such as measures for inheritance (Briand and Wüst, 2002; Singh et al., 2010; Jabangwe et al., 2013).

Size Measures: Lines of Code (LOC) and Number of Classes (NC) were used to measure the size of C++ source code. LOC and Number of Subroutines were used to measure the size of VB source code.

Complexity Measures: The following measures were used to quantify the complexity of the C++ source code: Number of Methods per Class (NOM), Average McCabe Cyclomatic Complexity, Average Statements per Method and Average Block Depth. The following measures were used to quantify the complexity of the VB source code: Statements in Biggest Subroutine and Average Block Depth.

Herraiz and Hassan (2012) found that simple size and complexity measures, such as LOC and McCabe, quantify sufficient information from source code. We argue that adding more complexity and/or size measures to the list of measures would not provide significantly different information, because these measures quantify size and complexity properties that are sufficiently distinct from each other.

Table 2 and Table 3 summarize the definitions of the size and complexity measures extracted for C++ and VB source code[2].

Defect Data: A requirements analyst at the company helped with defect data extraction, and isolation of relevant defect data, as well as linking the defects to the correct software releases. Defects used in the study were those that were a result of a deficiency in the source code and required a solution to be implemented directly in the source code. The defects were also post-release defects, i.e., customer reported defects. All other defects, such as defects related to documentation, were excluded from the analysis. This inclusion or exclusion of certain defects was done to maintain consistency and ensure that all defects across all releases were linked to source code fixes.

[1] The tool can be downloaded from http://www.campwoodsw.com/sourcemonitor.html.

[2] Detailed definitions of the measures can be found in the Source Code Monitor Tool that can be downloaded from http://www.campwoodsw.com/sourcemonitor.html.

Table 2: Definitions for C++ Measures

Property	Metric	Definition
Size	LOC	Number of physical lines, excluding blank lines
Size	NC	Number of classes and "structs"
Complexity	NOM	Average number of methods per class
Complexity	Average Statements per Method	The total number of executable statements inside methods divided by the number of methods
Complexity	Average McCabe Cyclomatic Complexity	Average complexity of methods/functions per file
Complexity	Average Block Depth	The average nested block depth

Table 3: Definitions for VB Measures

Property	Metric	Definition
Size	LOC	Number of physical lines containing source code, i.e., excluding comments and blank lines
Size	Number of Subroutines	Number of Subroutines and functions
Complexity	Statements in Biggest Subroutine	The maximum number of executable statements in a subroutine or function
Complexity	Average Block Depth	The average nested block depth

Analysis of Defect Data: Defect data from each product was analyzed using descriptive statistics and graphical representations that provide an evolutionary view of the captured measures. This helped with exploring trends and patterns of measures reported over time, across releases.

Analysis of Source Code Measures: The color-coded schemes of heat maps were used to analyze the change patterns that occur for the measures, across releases, before, during and after the handover. Changes in the measures between releases are represented by the shade of a color, which indicates the size of change in relation to the largest change. Shades of the color blue indicate a decrease, red indicates an increase and white or no color indicates that there was a very small change or no change at all between releases in the measure(s). The darker the color the closer the change is to the largest change that occurred across releases for the specific measure. Fractal and evolution matrix diagrams (Mens and Demeyer, 2008), and heated-object diagrams (Diehl, 2007) are just some of the diagrams that also use a color-coded graphical representation depicting a product's evolution.

Moving-range charts are used to identify periods with the largest shifts, and the relation/links in the shifts between measures, across releases, before, during and after the handover. To ensure comparability between variables, we normalized the magnitude of change from one release to the next. This was done by computing the ratio of change between releases in relation to the total change in the variable. Moving-range charts are commonly used in the context of Six Sigma to investigate the stability of process performance measures (Cagnazzo and Taticchi, 2009). Petersen and Wohlin (2010) use them to analyze requirements inventories over time, to investigate periods of high and low inventories, and to understand the extent of shifts between high number and low number of inventories at different periods.

All authors of this paper individually and independently analyzed defect data, heat maps, and moving-range charts, taking into consideration the timelines of events as shown in Figure 1 and Figure 2, and then met for a discussion of their observations. There was a high level of agreement on the potential relations of the shifts in measures and the distribution of work as well as the handover period.

Subjective Views: Follow-up questionnaires were then sent out to senior company representatives, that were or are involved in both products, to confirm or reject our observations. The questionnaires consisted of open-ended and closed-ended questions. They were administered through email with the help of a requirements analyst at the case company.

Results and analysis of the data are presented in the next section.

6 Results and Analysis

6.1 Evolution of Product-A and Product-B

There are 13 releases of Product-A between 2005 and 2012. These 13 releases are Product-A's "evolutionary" releases, i.e., excluding releases that only contained defect fixes. In each release, at least 85% of the code (as measured in KLOC) is programmed in VB, the rest is in C++. The first release had approximately 247 KLOC (approximately, 35 KLOC in C++ and 212 KLOC in VB). The latest release has slightly less than 300 KLOC (approximately, 29 KLOC in C++ and 270 KLOC in VB).

Between 2001 and 2012 there were 27 "evolutionary" releases of Product-B. All core source code components for Product-B were programmed in C++. The first release had 133 KLOC, and the latest release contains approximately 255 KLOC programmed in C++.

The next subsections present the evolution of the internal and external measures for Product-A, followed by those for Product-B.

6.2 Product-A (RQ1–RQ3)

Overview of Changes

The analysis of the measures extracted from C++ and VB source code, from Product-A, was done separately. Figure 3 and Figure 4 show heat maps that visualize the change patterns across releases for C++ components and VB components, respectively. The rightmost column in each of the figures shows the percentage of new functionality added in relation to the total functionality added across releases, during the evolution of Product-A.

A zero in a column in the heat map figures indicates that there was no change in the particular source code measure from one release to another. For example, the only change in the Biggest Subroutines measure occurs between R9–R10. The measure does not change before R9 or after R10. A negative value indicates a decrease in the source code measure, and a positive value (value without a negative sign) indicates an increase in the measure.

Product Development Responsibility	No. of Employees	Other Events	Handover Period	Releases	LOC	NC	AvgStmts	AvgMcCabeCC	NOM	AvgDepth	% of New Func.
Site-Alpha	Site-Alpha: 2 Dev and 1 Architect Site-Beta: 2 Dev			R1-R2	3	0	-0.18	0	0	0	10%
				R2-R3	-9773	-6	-4.77	0.8	0.02	0.29	8%
			Handover	R3-R4	0	0	0	0	0	0	7%
				R4-R5	0	0	0	0	0	0	2%
				R5-R6	0	0	0	0	0	0	2%
Site-Beta	Site-Alpha: 1 Architect Site-Beta: 1 Dev and 1 Dev/RA	Switch VCS Tool		R6-R7	7	0	0	0	0	0	2%
				R7-R8	25	0	0	0	0	0	5%
				R8-R9	3111	5	-2.31	0.2	-0.05	0.25	29%
				R9-R10	364	0	0.29	0	0	0.01	14%
				R10-R11	1	0	0	0	0	0	7%
				R11-R12	14	0	0	0.1	0	0.01	8%
				R12-R13	0	0	0	0	0	0	5%

Fig. 3: Product-A (C++) Heat Map

The unit of measure is the countable unit for each source code measure, e.g., LOC, except for percentage of new functionality which is in percentage.

Product Development Responsibility	No. of Employees	Other Events	Handover Period	Releases	LOC	No. of Subroutines	Biggest Subroutines	Avg Depth	% of New Func.
Site-Alpha	Site-Alpha: 2 Dev and 1 Architect Site-Beta: 2 Dev			R1-R2	-6201	-306	0	0	10%
				R2-R3	33909	890	0	-0.11	8%
			Handover	R3-R4	0	0	0	0	7%
				R4-R5	0	0	0	0	2%
				R5-R6	0	0	0	0	2%
Site-Beta	Site-Alpha: 1 Architect Site-Beta: 1 Dev and 1 Dev/RA	Switch VCS Tool		R6-R7	629	0	0	0	2%
				R7-R8	11260	196	0	-0.02	5%
				R8-R9	7986	175	0	0.03	29%
				R9-R10	6290	133	186	0.01	14%
				R10-R11	4309	55	0	0.1	7%
				R11-R12	619	-21	0	0.01	8%
				R12-R13	0	0	0	0	5%

Fig. 4: Product-A (VB) Heat Map

The unit of measure is the countable unit for each source code measure, e.g., LOC, except for percentage of new functionality which is in percentage.

The releases under the handover period indicated in Figure 3 and Figure 4 are releases that were actually made available during the handover activities. For example R2–R3 is not under the handover period highlighted in 3 and Figure 4 because R2 was not released during the handover period. This visualization approach is used because we do not include intermediate releases in our analysis, i.e., releases with only correctional packages. Besides, the heat maps are used to show the change of values from one release to another.

Each of the size and complexity measures is using a distinct method for quantifying source code properties. There are significant changes in all measures during similar periods, during Product-A's evolution. For example, R8–R9 and R9–R10 in both Figure 3 and Figure 4.

Two other observations can be made from the heat maps, before, during and after the handover period. First, during the handover period, size and complexity measures do not change between releases. See R3–R4, R4–R5 and R5–R6 in both Figure 3 and Figure 4. During this time actual development is also distributed between sites. The other observation is that the most significant changes in source code measures, for both Product-A's C++ and VB components, occur for the periods R2–R3 and R7–R8 onwards.

For R2–R3, during distributed development, source code size for C++ significantly decreases, whilst methods get larger and more complex on average (see R2–R3 in Figure 3). On the other hand, for VB source code, size increases but the complexity decreases (see R2–R3 in Figure 4). These patterns indicate refactoring activities and the moving of some functionality from C++ to VB source code. This was corroborated during the follow-up questionnaires from company representatives.

For C++ and R8–R9 onwards (see Figure 3), there are increases in size and complexity measures, LOC, NC, Average McCabe Cyclomatic Complexity and Average depth. For VB and R7–R8 onwards (see Figure 4), there are increases in size and complexity measures, LOC, Number of Subroutines, Biggest Subroutines and Avg Depth. These increases occur over a year after the handover when work is collocated at Product-A's Site-Beta.

An analysis of the new functionality added to Product-A across releases (the rightmost column in Figure 3 and Figure 4), shows that the largest changes of new functionality added across releases occur during corresponding periods when there are large changes in size and complexity measures. For the column with the new functionality added the color blue indicates periods with the smaller changes from one release to the next and the larger changes are indicated by the color red. For example, the releases with the largest overall increases in measures (R8–R9 and R9–R10) are the releases with the highest numbers of new functionality in comparison to other releases. Thus, the increase in size and complexity from R7–R8 onwards, could be linked to the comparatively large addition of new functionality.

Furthermore, an analysis of the defect inflow reveals that there was quite a uniform inflow of defects across releases, see Figure 5. Due to privacy and confidential reasons, we are not permitted to reveal the absolute values for the defect data. However, the variability and trend of the defect data shown in the figures is more important than the absolute values. This is because it shows the pattern in inflow of defects/problems (linked to the source code) and how it changes over time in relative terms, across multiple releases. The defect inflow during and after distributed development does not differ significantly. A similar observation can be made for the defect inflow before and after the handover period.

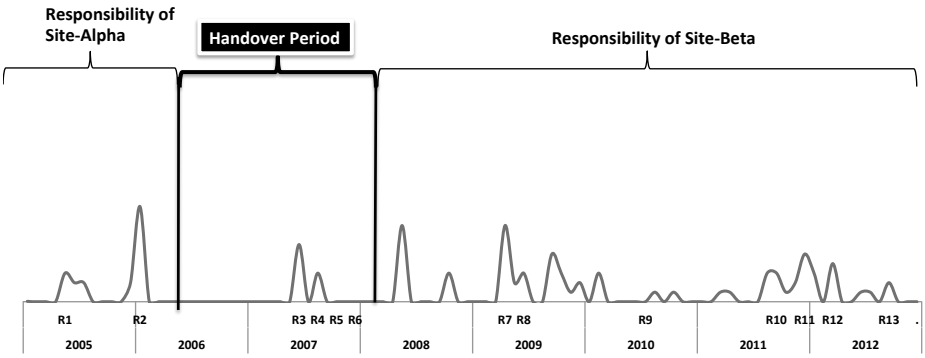


Fig. 5: Defect Data for Product-A

However, there are observable shifts in the internal and external product measures across releases. Visualizing these shifts over time, facilitates with comparing the magnitude of shifts before and after distributed work and at different handover periods, i.e., before, during and after the handover. This in-turn helps with understanding the extent of the influence that distributed development and the handover activities may have in the shifts. A visualization of these shifts, using moving-range charts, is presented in the next subsection.

Moving-Range Analysis

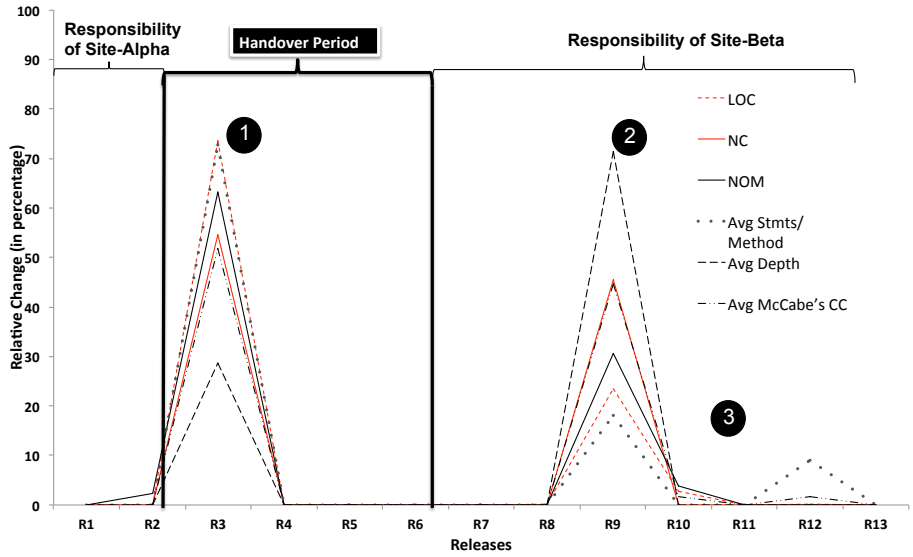
Moving-range charts for Product-A are shown in Figure 6. Releases are on the x-axis and percentages of change per release are on the y-axis. Figure 6(a) and Figure 6(b) show the moving-range chart for the internal product measures, i.e., size and complexity measures, for Product-A's C++ components and VB components, respectively. The relative change is in percentage of the total amount of changes. For example, if in total LOC in the complete time period has been changed by X, the percentage of LOC changes of that for each R1 to R27 is shown).

Figure 6(c) shows the moving-range chart for the external product measures, i.e. functionality added per release, time between releases and number of defects per release. Due to limited data available in the configuration management system we were unable to isolate the functionality added into each source code component. As a result, the visualization of the functionality added is performed at the product level.

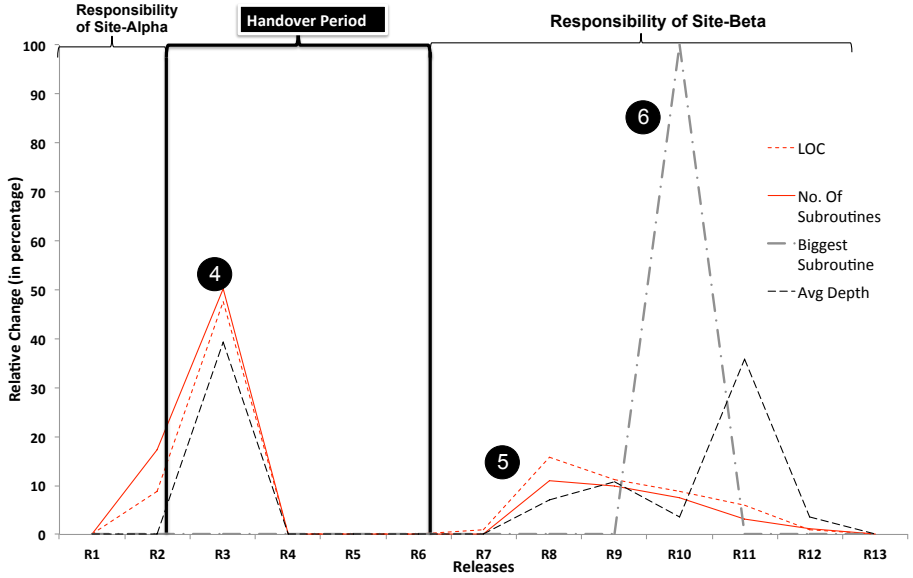
For C++, Figure 6(a) shows that the periods with the largest changes in size and complexity occur for R2–R3, indicated by ❶ and for R8–R9, indicated by ❷. R2–R3 is at the beginning of the handover and during distributed development. R8–R9 is almost two years after the handover period and during collocated development. There are no visible shifts between R4–R8 for C++. The other noticeable shifts occurs for R11–R12, indicated by ❸ in Figure 6(a). However, the shift is quite small in comparison to the shifts that occur for R2–R3 and R8–R9.

For the VB components, the largest changes occur at similar periods as for the C++ components. These periods are indicated in Figure 6(b) by ❹ and ❻. Another noticeable shift for size and complexity of VB components occurs for R7–R8, indicated by ❺, which is over a year after the handover and period of distributed development. However, this shift is small in comparison to those that occur for R2–R3 and R8–R9.

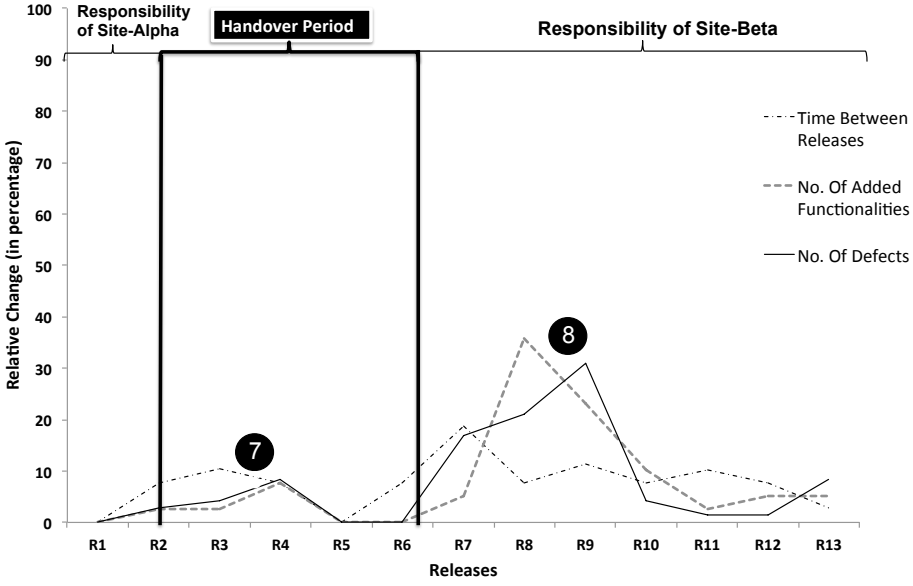
Apart from other maintenance activities such as refactoring activities, the shifts that occur during the handover for R2–R3, and almost two years after the handover for R8–R9, can be attributed to the evolutionary characteristics of the product. Figure 6(c) shows that there are visible shifts in all external product measures during the periods with large shifts in source code measures for VB and/or C++ components. Notably, there are also large shifts in the number of new functionality added, indicated by ❼ and ❽, that also occur during the same periods as the large shifts that are visible in the size and complexity measures for the VB and C++ components.



(a) Internal Product Measures (C++)



(b) Internal Product Measures (VB)



(c) External Product Measures

Fig. 6: Product-A: Moving-range Charts

6.3 Product-B (RQ1–RQ3)

Overview of Changes

A heat map that visualizes the change patterns in the size and complexity measures for Product-B's components is shown in Figure 7.

Product Development Responsibility	No. of Employees	Other Events	Handover Period	Releases	LOC	NC	Avg Stmts	Avg Complexity	NOM	Avg Depth	% of New Func.
Site-Alpha	Site-Alpha: 2 Dev and 1 Architect/RA Site-Beta: 3 Dev	Differing Processes between Sites		R1-R2	1320	2	0.1	0.02	0.02	0	8%
				R2-R3	12655	26	0.2	0.08	0.5	0.03	4%
				R3-R4	1544	4	0.1	0.01	-0.02	0	9%
				R4-R5	7492	17	-0.2	-0.05	0.38	0	4%
				R5-R6	8052	20	0.1	0.03	0.6	0.01	3%
				R6-R7	-6535	-18	0	-0.03	-0.56	0	3%
				R7-R8	8892	31	0.2	0.07	-0.09	0.01	1%
				R8-R9	0	0	0	0	0	0	3%
				R9-R10	1987	6	0	-0.01	-0.05	0	15%
				R10-R11	819	3	0	0	-0.01	-0.01	3%
				R11-R12	26032	83	-0.1	-0.01	0.08	-0.02	5%
				R12-R13	685	1	0	0.01	0.01	0	3%
				R13-R14	2270	6	0	0.01	-0.03	0	2%
				R14-R15	5816	10	0.1	0.01	0.03	0	1%
				R15-R16	197	0	0	0	10.56	0	3%
				R16-R17	3230	36	-0.4	-0.1	-11.07	-0.03	1%
				R17-R18	636	1	0	0	0.03	0	1%
				R18-R19	1623	0	0	0	0.02	0	2%
				R19-R20	425	0	0	0	0.02	0	3%
				R20-R21	16287	39	0.2	0.06	0.03	0.01	2%
	Site-Alpha: 1 Architect/RA Site-Beta: 3, gradually increasing to 6 Dev	Attempt to Merge Processes	Handover Period	R21-R22	3978	7	0.1	0.02	0.04	0.01	5%
				R22-R23	3813	2	0.1	0.03	0.02	0.01	4%
Site-Beta	Site-Alpha: 1-3 Dev (as needed) and 1 Architect/RA Site-Beta: 3 Dev	Abandon Merge of Processes		R23-R24	12426	11	-0.3	-0.07	0.47	-0.01	1%
		Switch VCS Tool		R24-R25	-3297	9	0.4	0.1	-0.48	0	3%
				R25-R26	10687	22	0.2	0.05	0.13	0.02	3%
				R26-R27	1539	3	0	0.01	0	0	3%

Fig. 7: Product-B Heat Map
The unit of measure is the countable unit for each source code measure, e.g., LOC, except for percentage of new functionality which is in percentage.

For Product-B, there was distributed development before, during and after the handover of project management responsibilities. Three observations can be made from the heat map. First, there are increases in size and complexity measures for the early product releases (for example, see R2–R3 in Figure 7). Second, there are increases in size and complexity measures during the handover period (see R20–R21, R21–R22 and R22–R23 in Figure 7). The third observation is the decrease in complexity measures for the release directly after the handover period (see R23–R24 in Figure 7).

Overall, the increases in the size and complexity measures for Product-B, can be linked to the addition of new functionality. An analysis of the functionality added per release (rightmost column in Figure 7), shows that releases with more new func-

tionality added have high increases in size and complexity measures. For example, more functionality is added to releases during the handover period (R20–R21, R21–R22, and R22–R23) than to most releases before (R12–R20) or after the handover (R23–R26). The decrease in complexity measures directly after the handover visible in R23–R24 for the measures Average Stmts, Average Depth, and Average McCabe Cyclomatic Complexity, is an indicator of refactoring activities, since it occurs in a period with low %-age of added new functionality.

Figure 8 shows the number of defects reported per release for Product-B. Due to privacy and confidential reasons, we are not permitted to reveal the absolute values for the defect data. However, the variability and trend of the defect data over time is of more importance. There is an observable higher inflow of defects before the handover, than during and after the handover. This high inflow occurs during early product development and when the product was still maturing. After R14, the defect inflow is more uniform and on a lower level. Although new functionality has been added continuously during and after the handover, the defect inflow seems unaffected by the handover.

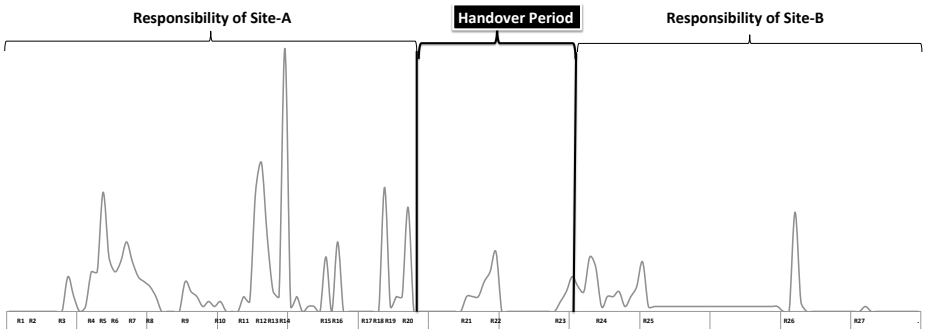


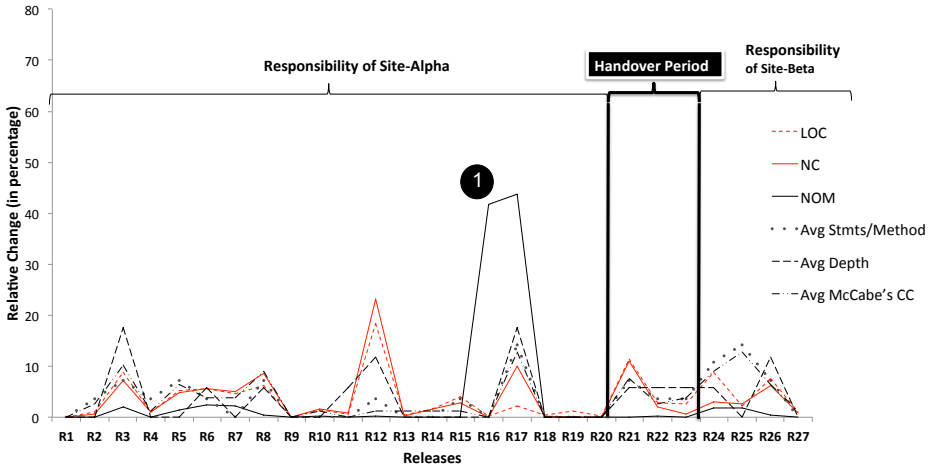
Fig. 8: Defect Data for Product-B

There are observable shifts without clear patterns in internal and external product measures for Product-B. Similar to the analysis for measures for Product-A, moving-range charts are also used to investigate the magnitude of the shifts in measures for Product-B.

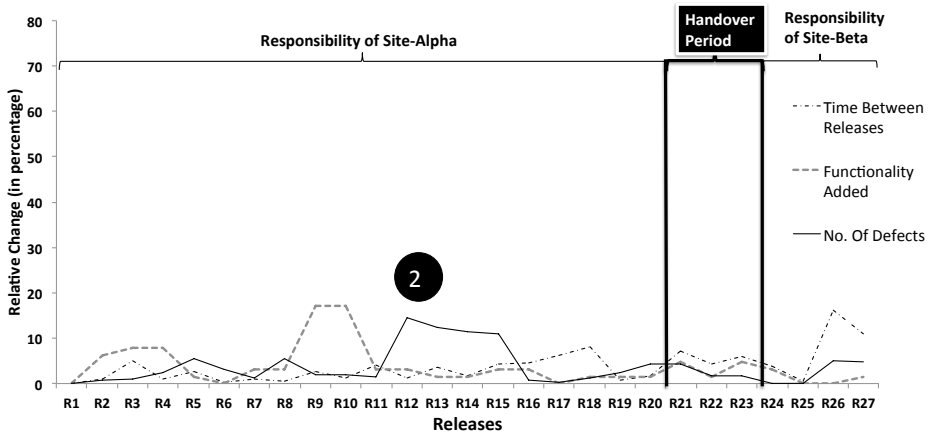
Moving-Range Analysis

Moving-range charts for Product-B are shown in Figure 9. Releases are on the x-axis and percentages of changes per release are on the y-axis. The relative change is in percentage of the total amount of changes from the first to the last release. Figure 9(a) shows the moving-range values for the internal product measures, i.e. size and complexity measures. Figure 9(b) shows the moving-range values for the external product measures, i.e. functionality added per release, time between releases and number of defects per release. The main changes in the figures are indicated by ❶ in Figure 9(a), and ❷ in Figure 9(b).

Figure 9(a) shows that the magnitude of change of the internal product measures did not significantly vary across releases. The most notable changes occur for releases



(a) Internal Product Measures



(b) External Product Measures

Fig. 9: Product-B: Moving-range Charts

R15–R16 and R16–R17 for the complexity measure NOM, indicated by ❶. After this period the magnitude of the shifts in all of the complexity and size measures, does not vary significantly during or after the handover period. This is also the case for the external product measures (see Figure 9(b)). The period of heavy changes in defect inflow for releases R12–R15, indicated by ❷ seem to be an effect of the large changes in new functionality added for releases R9–R10 (the dashed line in Figure 9(b) just before ❷). Thus, the difference in the shifts in the source code measures and the external product measures that occur before, during and after the handover across releases, in the distributed development setting, seem negligible.

In the next subsection, we provide a cross-case comparison of the two products, and present our overall findings so as to address research questions RQ4 and RQ5.

6.4 Overall Observation of Change Patterns and Shifts (RQ4 and RQ5)

Overall, there are no observable indications that the GSD setting that involved distributed development and the handover of project management responsibilities from one site to another negatively affected quality during the evolution of either Product-A or Product-B. There are no observable effects on the rate of defect inflow. The change patterns and shifts in size and complexity measures from the source code can be linked to factors other than the distribution of work or the handover, for example increases in new functionality added and refactoring activities. Therefore, there is no discernible link between the shifts in both the source code measures and the external measures and the distribution of work or handover for either product.

Though we make similar observations from the two products, it is important to note that the two products have different evolutionary characteristics. Product-B has more releases than Product-A before the handover. Hence more new functionality is added to Product-B before the handover in comparison to Product-A. On the other hand, Product-A has more releases than Product-B after the handover. Another difference in evolution between the products is that Product-B has undergone more evolutionary changes, and a longer period of distributed development, than Product-A. It has twice as many releases over a longer time period in comparison to Product-A. Furthermore, Product-B has more variation in the change patterns for source code measures than Product-A (see heat maps in Figure 3, Figure 4 and Figure 7).

Thus there are not only evolutionary differences between the products. Superficially, they look very similar: same company, same sites, same domain, handover of product responsibilities from one site to another, distributed development before the handover, and approximately the same size of source code components that capture evolutionary changes. However, the products have the following differences: one product has source code components from two programming languages whereas the other product only has one, different histories in number of releases before and after the handover, in one product there was a shift from distributed to collocated development, the defect inflow for Product-A is much more steady and uniform than that for Product-B, and different evolution history in terms of both number of releases and change/shifts in source code measures, before, during and after the handover.

7 Discussion

The results of our case study suggest that it is possible to handover project management responsibilities between two geographically separated sites without it negatively affecting the quality of the source code for evolving systems. Size and complexity measures, as well as defect data, can be used as indicators of quality related concerns (Jabangwe et al., 2013); there was no observable variation in the measures or defect data that suggested a decline in quality. Albeit, project management responsibilities were at a single site at any given time, software development work was shared between two sites, particularly, before and during the handover period for both products. Nagappan et al. (2008) found that the organizational structure can influence quality. In our study, there was no discernible impact on quality, across releases, due to the shared development work between geographically dispersed sites. In addition there was no observable difference in quality between distributed development and collocated development. Thus, our findings do not corroborate findings by Ra-

masubbu and Balan (2007), but they are consistent with the findings by Spinellis (2006) and by Bird et al. (2009). Overall, our observations were also confirmed by company representatives for both products from both sites.

Success factors: Overall, the senior company representatives, for both products in our study, perceived the distribution of work and the handover to be a success. In our follow-up questionnaires, they identified the following factors as contributors to the success of the handover:

- Availability of skilled employees before, during and after the handover at sending and receiving sites.
- Well-designed software architecture.
- Use of face-to-face meetings for knowledge transfer activities.
- Early involvement of employees of the receiving site.
- Prolonged time of distributed development between the sending and the receiving site before the handover, and in-turn build-up necessary competencies at the receiving site.
- Nearshoring, i.e., offshoring to a close destination.
- Low variation of product complexity and size across releases.
- Early planning and dissemination of the handover activities.
- Effective communication and coordination practices between sites.
- Skilled and competent employees at sites involved in distributed development and handover activities.
- Sufficient product documentation.
- Support from the sending site during and after the handover of project management responsibilities.

It is, however, important to understand the reasons why our study results may seem to contradict, or corroborate, findings from other studies. The above enabling factors as well as other factors that were identified from company documentations on the handover activities are discussed further below. Since we investigate the handover period similar to how previous studies investigated software transfers, software transfer critical factors described by Šmite and Wohlin (2011), i.e., product, people and process, are used to guide the discussion.

Product Factors: A complex and poorly designed product architecture can negatively impact the rate of learning for employees at the receiving site (Šmite and Wohlin, 2011). In our study, both products are medium-size and, as shown in our quantitative analysis results, their complexity does not vary significantly during product evolution before, during and after the handover. This was also consistent with the views from the senior company representatives. The company representatives also pointed out that the "relatively good" architecture of the product allowed for fast and efficient knowledge transfer.

When investigating the effects of a handover, we could not corroborate the findings of a negative effect on quality as a result of a transfer reported by Jabangwe and Šmite (2012). The major reason is that a transfer is much more challenging to execute than the handover observed in our study. In addition, the product used in the study by Jabangwe and Šmite is much larger, and, thus, it is likely to be more complex and more defect-prone, than each of the two products in our study. The product they used has releases that are three to six times larger in size, measured in LOC, than the product releases we use in our study. We also believe that the product used in the study by Jabangwe and Šmite (2012) caters to a market that

is more volatile in comparison to each of the two products used in our study. This would make their product more change-prone, and hence more defect-prone.

Process Factors: Project costs can be linked with project risks (Lagerström et al., 2012). Early planning can help with identifying potential risks, and planning mitigation strategies that can substantially reduce both expected and unexpected costs (Šmite and Wohlin, 2011). In addition, a clear and transparent documentation of the transfer plan also plays an important role in executing a successful transfer (Šmite and Wohlin, 2011). Project managers for both products in our study indicated that the handover was planned and disseminated early, and the handover process was well-established and documented, and transparent to both the receiving and sending sites. The company also used a stepwise and gradual process to relocate project management responsibilities. This is a similar process to the gradual software transfer described by Wohlin and Šmite (2012); were the project management responsibilities were relocated rather than the development and maintenance work as would be the case for transfers.

Given that the company had been involved in planning the handover for a long time before it began, it is possible that the company had avoided adding complex functionality during the period. However, we do not have sufficient data to support this conjecture.

Company representatives also recognized the importance and the positive effect that face-to-face meetings during knowledge transfer activities had on the success of the handover activities and distributed development. Similar observations have been reported in numerous GSD studies (Nidhra et al., 2013; Griffith and Sawyer, 2006). Project managers for the receiving sites for both products indicated that the handover of vital product aspects was primarily done during face-to-face meetings. Employees from Product-A's Site-Beta and Product-B's Site-Beta were involved in several trips to Sweden for knowledge transfer. Furthermore, managers perceived knowledge transfer activities as successful because after a certain period after the handover, support from Sweden was not needed and neither was it necessary.

People Factors: Familiarity and experience with a product can have an impact on performance (Huckman et al., 2009). Early involvement in development activities helped the receiving sites in building up necessary skills and competencies, and in-turn mitigate the risk of a decline in quality after the handover of responsibilities. The periods of involvement of the sites is highlighted in Figure 1 and Figure 2.

Coordination and communication are vital for running a successful project (Brooks, 1995; Bird et al., 2009). In general, the project managers for both products, were in agreement on the importance and effectiveness of cross-site communication and coordination during the handover activities. Software development teams at both the sending and receiving sites, for both products in our study, were small and stable. In addition, the close proximity of the two sites, which can be referred to as nearshoring (Carmel and Tjia, 2005), mitigated communication problems or delays that may arise from, for example, temporal distance.

At the time of the handover, employees at the receiving site for both products had skilled software engineers. To mitigate risks, there was also prolonged support to the receiving site from the sending site's management and development teams. These two factors were noted as contributors to the success of the handover from managers from both products.

In summary, the two products met many of the characteristics that were noted by Šmite and Wohlin (2011) for products that are good candidates for relocating

work between sites. The receiving site had product experience and had ample time to build up competencies in preparation for the handover. The products also had a well-designed architecture, low variability in complexity and size properties across releases, they were mature, non-volatile and had sufficient documentation available. All these factors helped the receiving site in taking over product responsibilities.

There were limited risks associated with moving management responsibilities in the two studied cases due to context factors such as small sized teams and medium sized products. Changes in these context variables could have resulted in an implication on the reflections discussed. However, none of the respondents directly pointed out team size as a contributing factor for the success of the handover. Research shows that coordination between large teams can result in complex communication networks, and this complexity can manifest into the source code, and in turn may have a detrimental impact on product quality (Herbsleb and Grinter, 1999). Thus, this makes the case that we have presented interesting as it shows how quality for medium sized products may vary in setups with small teams.

However, the success factors of this handover case study might give some useful input for companies considering a transfer. Structuring or planning a transfer more like a handover might mitigate many of the typical transfer related risks. For example, a company may ensure that there is a long period of distributed development work between sites, during which the receiving site has time to build-up necessary skills and competencies.

8 Validity Threats

Threats to validity (Robson, 2011; Wohlin et al., 2012) are discussed below.

Construct Validity: Construct validity pertains to measuring what is expected to be measured (Robson, 2011). The data we used was from two real commercial software systems that are currently on the market, and we believe, the following two strategies mitigated the likelihood of measurement errors. Identification and collection of releases and source code artifacts was done with the help of architects at the company that had extensive knowledge on the evolution of both products. Extraction of all source code measures was done using an automated tool (See Section 5). We believe, these two strategies mitigated the likelihood of measurement errors.

Static analysis tools that extract source measures can have different methods and approaches for quantifying certain properties (Lincke et al., 2008). As a result, they may produce inconsistent values for measures. We address this issue by using the same tool for both products. This makes the measures that we use in the study comparable.

Internal Validity: Internal validity is concerned with the accuracy of the cause-effect relationships reported in a study (Robson, 2011). We make inferences, such as, refactoring as a reason for source code changes, and the lack of impact of transfers on quality. The inferences are, however, triangulated with data collected through questionnaires from company representatives that have extensive knowledge regarding the evolution of both products in this study.

Since there are very few employees involved in product development, we were unable to conduct a meaningful statistical analysis from the data collected through questionnaires. Nevertheless, this should not affect our results and findings, since the purpose of the questionnaires was not to explore or identify or extract new

information. The questionnaires were only used for the purpose of confirming or rejecting our observations. For this reason, we can not claim that the list of factors that contributed to the success of both handovers (see Section 7) is exhaustive.

Theoritical Validity: Pertains to confounding factors that may influence and affect a theory about a phenomenon or relationship between variables (Petersen and Gencel, 2013). Time delay of various development and maintenance processes, e.g., time between first defect reported for each release (which is related to variability of the intensity of system usage by the consumers), are factors that could influence defect inflow across releases. Unfortunately, at the time of the study we could not quantify or capture information about this confounding factor.

External Validity: External validity pertains to the extent of the generalizability of study results (Robson, 2011). The study presented is an embedded case study within the same company and the results presented are specific to the two products. However, providing a detailed description of the contexts of the products (see Section 3), helps in improving the study's external validity as pointed out by Petersen and Wohlin (2009). There are also many differences in the evolutionary characteristics of the two products (see Section 6.4) that strengthen external validity of our results.

The results we report are likely to be generalizable to companies that are planning to, or are involved in, transfers for medium-sized products. Besides, our study findings, e.g., the "success factors" discussed in Section 7, can be used to formulate valuable input for GSD related decision-making structures such as the one described by Šmite et al. (2013).

Additionally, analytical induction can help with understanding the extent of generalizability for studied cases (Wieringa, 2013). For example, there may be similarities in certain mechanisms with other case contexts, such as, size of development teams and the intention to relocate product management activities from one site to another. These similarities, as well as the differences are important to note so as to understand the extent of generalizability of the findings that are presented in this study. Understanding what did or did not work in a certain setting would be useful information as input in the decision-making for settings with similar characteristics. Based on this notion, our case description in Section 3 is detailed so as to improve the external validity of this study.

Conclusion Validity: Conclusion validity pertains to the accuracy of conclusions drawn from the cause-effect relationships reported in a study (Wohlin et al., 2012). This risk is reduced by using multiple researchers. The first author collected all raw data, and then all authors were involved in developing all diagrams. Actual analysis was done independently and then discussed. Conclusions were triangulated with subjective opinions from company representatives.

It is important to note that, in both cases studied, the size of the teams was small and the systems were medium-sized. In case a handover or transfer took place in a different context where many more teams and more complex products were involved, different effects might be seen.

9 Conclusion

In the present embedded explanatory case study we show that software project management responsibilities can be handed over between two geographically dispersed sites without it negatively impacting quality. We also show that development work

can be carried out between geographically dispersed sites and increase the chances of realizing potential benefits of GSD, such as reducing costs. We found no observable impact of distributed development on quality, which is consistent with the findings by Spinellis (2006) and by Bird et al. (2009). We could not corroborate findings of a negative effect on quality as a result of distributed development reported by Ramasubbu and Balan (2007). Investigation of quality is done using source code measures and defect data from two medium-size commercial products. The two products have different evolution characteristics and we found no indication of an impact on quality as a result of handover of project management responsibilities or distributed development.

We identified factors that contributed to the success of the handover for both products. These factors were previously identified as critical factors for software transfers by Šmite and Wohlin (2011). Thus our study can be considered as an indirect validation of the factors. Development teams were small at receiving and sending sites, thus mitigating communication and coordination overhead-related risks. Due to early engagement in product development, the employees from the receiving site had time to gain experience with the product and time to build up necessary skills and competencies. The products had a well-designed architecture. The sites involved had a close proximity, and they also had small software development teams, which decreased the communication and coordination overhead within and across teams. Face-to-face meetings were effective for knowledge transfer activities. Handover activities, were well-documented, planned and disseminated in advance. Support provided by the sending site to the receiving site was also noted as a contributor to the success of the handover.

Representatives from the case company identified the aforementioned factors as contributors to the success of the handover activities. For future work, we propose that a similar investigation should be conducted in a separate case; preferably in a case with different findings on the effect of handover activities on quality and compare the set of factors that we found. This is because if the set of factors were found in for example, a successful handover context and not a failed handover context, it would further substantiate if the factors contributed to success.

Visualization techniques used in this study were chosen due to that they provide simple visual images that conceptualize changes in data over time. This in turn helped with identifying patterns and trends, and gaining insights, in data across releases (i.e., during product evolution). Hence, this study shows how visualization mechanisms applied to multiple subsequent releases, and capturing events that occur during evolution, can be a useful tool for identifying factors that affect quality during evolution. The information produced (for example, effects of shifting development settings and distributed development, success factors, etc.) from the approach can potentially be invaluable input in the decision-making process for planning and managing future GSD projects.

Replicating our study in different contexts may provide valuable input on how changing context variables affects the quality (e.g., the relative magnitude of different product sizes or team sizes), having this information allows practitioners to make informed decisions on whether to perform a handover and what to expect for their particular context. In that sense, our study provides a first step towards gaining this understanding. In the event that similar studies as the one presented in this paper are performed, researchers should try and include pre-release defects (linked to a

problem in the source code) in the analysis. This information may provide a more detailed representation of the source code quality at different periods in time.

The body of knowledge on GSD settings and their impact on quality is scarce. Future studies should consider conducting a similar analysis on larger and more volatile (commercial) products that are change-prone. Such products are reported to be more susceptible to quality-related issues during evolution (Lu et al., 2012). A similar empirical study on such products would help with confirming or refuting this conjecture. There is also still a need for other investigations conducted at a similar or more extensive magnitude of other GSD settings, for example settings that involve software transfers.

Acknowledgements This work was funded by the Swedish Knowledge Foundation under the research grant 2009/0249. We thank Professor Claes Wohlin and Dr. Darja Šmite, at Software Engineering Research Lab (SERL), for their valuable comments on the paper.

References

Aspray, W., Mayadas, F., and Vardi, M. Y. (2006). *Globalization and offshoring of software: A report of the ACM job migration task force*. New York, USA.

Bansiya, J. and Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, 28(1):4–17.

Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751–761.

Bird, C., Nagappan, N., Devanbu, P., Gall, H., and Murphy, B. (2009). Does distributed development affect software quality? An empirical case study of Windows Vista. In *Proceedings of the 31st international conference on software engineering*, pages 85–93.

Briand, L. and Wüst, J. (2002). Empirical studies of quality models in object-oriented systems. *Advances in computers*, pages 97–166.

Brooks, Jr., F. P. (1995). *The mythical man-month (anniversary ed.)*. Addison-Wesley Longman Publishing, Boston, USA.

Cagnazzo, L. and Taticchi, P. (2009). Six sigma: A literature review analysis. In *Proceedings of the international conference on e-activities and information security and privacy*, pages 29–34.

Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Prentice Hall PTR, New Jersey, USA.

Carmel, E. and Tjia, P. (2005). *Offshoring information technology: Sourcing and outsourcing to a global workforce*. Cambridge University Press, Cambridge, UK.

Conchúir, E. O., Holmström, H., Ågerfalk, P. J., and Fitzgerald, B. (2006). Exploring the assumed benefits of global software development. In *Proceedings of the 1st international conference on global software engineering*, pages 159–168.

Diehl, S. (2007). *Software visualization – Visualizing the structure, behaviour, and evolution of software*. Springer-Verlag Berlin Heidelberg, New York, USA.

Griffith, T. L. and Sawyer, J. E. (2006). Supporting technologies and organizational practices for the transfer of knowledge in virtual environments. *Group decision and negotiation*, 15:407–423.

Herbsleb, J. D. and Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on software engineering*, pages 85–95.

Herraiz, I. and Hassan, A. E. (2012). *Making software – What really works, and why we believe it*, chapter Beyond lines of code: Do we need more complexity metrics?, pages 125–141. O'Reilly Media.

Huckman, R. S., Staats, B. R., and Upton, D. M. (2009). Team familiarity, role experience, and performance: Evidence from Indian software services. *Management science*, 55(1):85–100.

ISO/IEC/IEEE-24765 (2010). Systems and software engineering – Vocabulary. International organization for standardization.

Jabangwe, R., Börstler, J., Šmite, D., and Wohlin, C. (2013). Empirical evidence on the link between object-oriented measures and external quality attributes: A systematic literature review. *Accepted for publication at Empirical software engineering*.

Jabangwe, R. and Šmite, D. (2012). An exploratory study of software evolution and quality: Before, during and after a transfer. In *Proceedings of the 7th IEEE international conference on global software engineering*, pages 41–50.

Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C., and Tsirakis, N. (2010). Code quality evaluation methodology using the ISO/IEC 9126 standard. *International journal of software engineering and applications*, 1(3):17–36.

Lagerström, R., Würtemberg, L. M., Holm, H., and Luczak, O. (2012). Identifying factors affecting software development cost and productivity. *Software quality control*, 20(2):395–417.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68:1060–1076.

Lincke, R. d., Lundberg, J., and Löwe, W. (2008). Comparing software metrics tools. In *Proceedings of the international symposium on software testing and analysis*, pages 131–142.

Lu, H., Zhou, Y., Xu, B., Leung, H., and Chen, L. (2012). The ability of object-oriented metrics to predict change-proneness: A meta-analysis. *Empirical software engineering*, 17:200–242.

Mens, T. and Demeyer, S. (2008). *Software evolution*. Springer-Verlag Berlin Heidelberg, New York, USA.

Mockus, A. and Weiss, D. M. (2001). Globalization by chunking: A quantitative approach. *IEEE software*, 18:30–37.

Nagappan, N., Murphy, B., and Basili, V. (2008). The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th international conference on software engineering*, pages 521–530.

Nidhra, S., Yanamadala, M., Afzal, W., and Torkar, R. (2013). Knowledge transfer challenges and mitigation strategies in global software development - A systematic literature review and industrial validation. *International journal of information management*, 33(2):333–355.

Nurdiani, I., Jabangwe, R., Šmite, D., and Damian, D. (2011). Risk identification and risk mitigation instruments for global software development: Systematic review and survey results. In *Proceedings of the 6th international conference on global software engineering workshop*, pages 36–41.

Petersen, K. and Gencel, C. (2013). Worldviews, research methods, and their relationship to validity in empirical software engineering research. In *Joint conference of the 23rd International workshop on software measurement and International conference on software process and product measurement*, pages 81–89.

Petersen, K. and Wohlin, C. (2009). Context in industrial software engineering research. In *Proceedings of the 3rd international symposium on empirical software engineering and measurement*, pages 401–404.

Petersen, K. and Wohlin, C. (2010). Software process improvement through the lean measurement (SPI-LEAM) method. *Journal of systems and software*, 83(7):1275–1287.

Ramasubbu, N. and Balan, R. K. (2007). Globally distributed software development project performance: An empirical analysis. In *European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 125–134.

Robson, C. (2011). *Real world research.* John Wiley & Sons, West Sussex, UK, 2nd edition.

Runeson, P., Höst, M., Rainer, A., and Regnell, B. (2012). *Case study research in software engineering.* John Wiley & Sons, New Jersey, USA.

Singh, Y., Kaur, A., and Malhotra, R. (2009). Comparative analysis of regression and machine learning methods for predicting fault proneness models. *International journal of computer applications in technology*, 35(2):183–193.

Singh, Y., Kaur, A., and Malhotra, R. (2010). Empirical validation of object-oriented metrics for predicting fault proneness models. *Software quality journal*, 18(1):3–35.

Spinellis, D. (2006). Global software development in the freeBSD project. In Kruchten, P., Hsieh, Y., MacGregor, E., Moitra, D., and Strigel, W., editors, *Proceedings of the international workshop on global software development for the practitioner*, pages 73–79.

Verner, J., Brereton, O., Kitchenham, B., Turner, M., and Niazi, M. (2012). Systematic literature reviews in global software development: A tertiary study. In *Proceedings of the 16th international conference on evaluation assessment in software engineering*, pages 2–11.

Šmite, D. and Wohlin, C. (2011). Strategies facilitating software product transfers. *IEEE software*, 28(5):60–66.

Šmite, D. and Wohlin, C. (2012). Lessons learned from transferring software products to india. *Journal of software: Evolution and process*, 24(6):605–623.

Šmite, D., Wohlin, C., Aurum, A., Jabangwe, R., and Numminen, E. (2013). Offshore insourcing in software development: Structuring the decision-making process. *Journal of systems and software*, 86(4):1054 – 1067.

Šmite, D., Wohlin, C., Feldt, R., and Gorschek, T. (2008). Reporting empirical research in global software engineering: A classification scheme. In *Proceedings of the 3rd international conference on global software engineering*, pages 173–181.

Šmite, D., Wohlin, C., Gorschek, T., and Feldt, R. (2010). Empirical evidence in global software engineering: A systematic review. *Empirical software engineering*, 15(1):91–118.

Wieringa, R. (2013). Case study research in information systems engineering: how to generalize, how not to generalize, and how not to generalize too much. In *25th International Conference on Advanced Information Systems Engineering, CAiSE*, pages xii–xii.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering: An introduction*. Springer-Verlag Berlin Heidelberg, New York, USA.

Wohlin, C. and Šmite, D. (2012). Classification of software transfers. In *Proceedings of the 19th Asia-Pacific software engineering conference*, volume 1, pages 828–837.