An Exploratory Study of Software Evolution and Quality: Before, During and After a Transfer

Ronald Jabangwe
Blekinge Institute of Technology
Karlskrona, Sweden
Ronald.Jabangwe@bth.se

Darja Šmite
Blekinge Institute of Technology
Karlskrona, Sweden
Darja.Smite@bth.se

***Abstract*—In the light of globalization it is not uncommon that different project teams in different locations develop software products during evolution. However, empirical evidence that demonstrates the effect of changing project team members on software quality is scarce. In this paper, we investigate quality of a software product, across subsequent software releases, that was first developed in one location of a company, then jointly with an offshore location of the same company, and finally transferred offshore. To get a better understanding multiple data sources are used in the analysis: qualitative data, consisting of interviews and documentation, and quantitative data, consisting of release history and defect statistics. Our findings confirm an initial decline in efficiency and quality after a transfer, and highlight the areas of concern for companies that are considering transferring their product development from experienced teams to those having limited or no previous engagement with the product.**

Keywords-component; Global Software Development; Software Transfer; Software Evolution; Software Quality

I. INTRODUCTION

Software is bound to go through numerous changes due to variances of environmental variables, for example changes in user requirements, in which it operates. As a result software evolution is an inevitable phenomenon. Software evolution refers to the dynamic change of characteristics and behavior of the software through, for example, maintenance activities or implementation of enhancements, over time [1]. The phenomenon of software evolution was first observed in 1970 and subsequently led to the development of seven laws of software evolution between 1974 and 1985 [5–7]. The influential and compelling work by Belady and Lehman [6] on software evolution resulted in an increase in popularity of the topic as a research field [8].

This paper describes the findings from a retrospective analysis of evolution of a mature software-intensive product from a large telecommunication company. The purpose of this exploratory study is to assess software quality across releases as the software evolves, building upon the second law of evolution, the law of increasing entropy/complexity [6], and the seventh law, the law of declining quality [5, 7], which hypothesizes a link between software evolution and the potential decline in quality. Additionally, we study the effect of challenges introduced by global software development (GSD), as another complication in software evolution.

The company studied is involved in GSD, in which development occurs in various settings, such as offshore and distributed development. The role of GSD and its impact on quality has attracted recent interest in the software engineering community, but only a few empirical studies are published [10, 11]. While the early days of GSD as a field provided a mix of experiences with offshore development, it has been advised to distinguish unique settings and scenarios according to the differing and distinct characteristics they embody [15]. An example of a classification of different settings can be found in the work of Šmite and Wohlin [9]. In our case, it is not uncommon that existing development work at the company studied is transferred for further development or maintenance to its offshore sites. Relocation of software development work, which is also called a software transfer leads to a full exchange of experienced developers with new developers who may have limited or no previous engagement with the product [9]. While maintenance research suggests that new personnel is less efficient as they climb the learning curve, we propose that the effect of a software transfer has a similar implication with much more significant impact.

The remainder of the paper is organized as follows. Section II outlines research related to our study followed by a description of both the context and the research methods used in this study, in

Section III. Section IV discloses results from the quantitative and qualitative analysis, followed by a discussion highlighting findings from our study regarding, software evolution, quality and transfer, in Section V. Validity threats and study limitations are discussion in Section VI. The paper ends with the conclusion and future work, in Section VII.

II. Related Work

Research related to the topic of this paper is threefold. First of all, we refer to research on how software quality can be explored, secondly, to the relation of software evolution and quality, and thirdly, to research on the influence of offshore development and software transfers on quality.

Objective data and measures (i.e. related to defect data) are often used to evaluate quality [1]. However, subjective views on quality can also be used for software quality assessments. Chang-Peng et al. [10] use both subjective views and objective data to investigate quality. The objective characteristics that they used to denote quality were the number of defects found and software complexity as an indirect measure of quality (calculated using measures, including well-known measures for object-oriented designs from Chidamber and Kemerer [11]). Complexity was also studied subjectively through perceptions of complexity expressed by developers. Similarly, Xenos and Christodoulakis [12] propose a method that can be used to measure perceived quality of both internal customers, employees that also act as customers, and external customers, end-users. In their study of 46 projects a positive correlation was found between objective measures obtained from the internal characteristics of the product, such as McCabe's cyclomatic complexity, and measures derived and based on customers notion of quality (obtained through surveys).

Practices that ensure that software is running as expected, such as fixing defects, are part of software maintenance [4]. Maintenance and evolution as terms are often used interchangeably in academia and industry [1, 2]. The Software Engineering Book Of Knowledge (SWEBOK) [3] lists evolution as a sub-knowledge area of maintenance. In this paper, we distinguish software evolution from pure maintenance activities to refer to the process of adding new features as well as correcting defects that result in a new software release that has distinguishing characteristics from the predecessor version.

Since the early work on software evolution by Lehman and Belady [6] more studies have been carried out on the relation of software evolution and defects. Different measures have been found to link software artifacts with observed or perceived quality. From the evolution perspective an important concept is code churn measures. These are a set of measures that by their definition attempt to capture and objectively measure changes over time of evolution of software artifacts (for example, deleted lines of code) [8,13]. Results from the study conducted by Nagappan et al. [13] show that, in the context of their study, code churn measures are positively correlated to defects. Defect reports are an essential concept for quality assurance purposes [1].

As software evolves, quality assurance requires accounting for the impact of development environment and processes [8]. One of the potentially influencing factors that are associated with quality concerns is globalization of software development. Different settings in GSD are associated with unique characteristics and consequently unique challenges such as different work practices, asynchronous work, and cultural differences [9]. These challenges can inhibit the realization of quality goals. Furthermore, changing from one setting to another can have an impact on quality, as in the case of software transfers when development of an evolving software is moved from one development team to another. Although there is little research focused on software product transfers, an empirical study suggests that transfers cause a decrease in development efficiency and harder to capture secondary negative effect on quality [14].

Motivated by the gaps in related work, this study reports on the changes in quality across different releases of a software product before, during and after a transfer. Investigation of the effects of transfer on software quality is an important contribution of this study to the body of knowledge on GSD. In contrast to many studies taking a static perspective, this study captures an evolutionary view on quality. In particular release history, defect reports and documentation are used to conduct software evolution analysis as suggested by [1]. Objective evaluation is further complemented by

subjective views obtained from developers, as suggested by [28].

III. Research Methodology

A. Purpose

The objective of this exploratory study is to evaluate software quality through objective measures and subjective views during software evolution in a GSD context. The study is conducted in a company that develops software-intensive products globally for the telecommunications market. Though the product used in the study was transferred from one location to another within the same company during its evolution, the study does not dwell on the details of the transfer itself but rather on the product quality and evolution before, during and after the transfer. This study is driven by the following research question:

RQ: *How does software quality vary in a GSD setting that involves a transfer?*

The objectives here are to explore the prevailing perception on quality across releases (to capture subjective views) and the variation of defects reported across releases (to capture the objective measures).

B. Case Description and Context

Research reported in this paper is an empirical single-case exploratory study, which is conducted according to recommendations from Runeson and Höst [20]. The case company is Ericsson, which is a large multinational corporation that develops software-intensive products catering for the global market. The company is selected based on availability and interest in the research in this area. Recently a number of products were transferred from one of the company's locations (Site-A) to different offshore sites, and the company was interested in understanding the impact of such changes. The company selected one particular product for these purposes. The product's development was initially carried out at Site-A in Sweden and then gradually transferred to another Ericsson location Site-B in India.

The product under study has a long history. Development of the software product commenced in 2001 at Site-A and the product was released to the market in 2007. Employees from Site-B were temporarily moved to Site-A primarily for practical training. The transfer from Site-A to Site-B was carried out and completed in 2009 at which point it was already a mature product. Details of the transfer can be found in [14] (in the article the transfer for this particular product is referred to as project B).

Šmite et al. [15] proposed a classification of GSD related empirical studies to help understand the context and the extent of applicability and generalizability of reported studies related to GSD. Table 1 shows how the study reported in this paper fits into the GSD field according to characteristics of GSD scenarios provided in [9] and the classification in [15].

TABLE I. Study Characteristics in GSD Context

Empirical Background	***Main Method***	Case Study
	Sub methods	Interviews, quantitative analysis of defect reports
	Empirical Focus	Empirically-based (exploratory)
	Subjects of Investigation	Practitioners
GSD Background	***Collaboration Mode***	Intra-organization/Offshore insourcing
	Approach and Type of Work	Single-site execution of software product development in Site-A, parts of which were further transferred from Site-A to Site-B resulting in distributed work, and then finally transfer of the remaining parts to Site-B, which resulted in the single-site execution Single site execution implies a sites responsibility of all product development activities
Study Background	***Focus of Study***	Software evolution and quality

C. Data Collection and Analysis

In empirical research such as case studies, triangulation is an approach that can be used to strengthen, and increase accuracy and validity of findings [16]. Data and methodological triangulation were thus used for this purpose. Qualitative analysis results were used to consolidate the results obtained from quantitative analysis.

The use of both qualitative and quantitative methods is also referred to as the mixed method approach [17]. The motivation for using this approach in this study is that it helps to understand the context in which the product was developed and to obtain quality aspects from different viewpoints (i.e. objective measures and subjective views) during the evolution of the product. Thus quality is analyzed using subjective views of those involved

with the product's development work, prior to the transfer, as well as using objective measures before and after the transfer.

1) Quantitative Data

Quantitative analysis involved defect data and product release history. A defect in this study is used to refer to any reported deficiency or imperfection or problem in the source code that resulted in the product producing results that deviated from the expected outcome, as defined in the product specifications, and as a result required a solution to be implemented directly into the source code, which is a slight modification of the two definitions from the International standard ISO/IEC/IEEE 24765[a] [4]. This includes defects found regardless of the software life cycle or phase (for example, before or after deployment at customer sites) or cause, severity, detection method (e.g. static analysis or during execution), type or solution method.

Defects reported between 2007 and 2011 were extracted from the company's database and used for the purposes of this study. Only code-related defects were considered, thus an initial defect analysis was conducted to identify specifically defects that were linked to a deficiency or imperfection or problem in the source code, hence excluding documentation and other defects that are irrelevant to the purpose of this study.

A test and verification expert at the case company assisted with defect data extraction, compilation and analysis. A series of meetings were held with the expert to discuss and to ensure that appropriate defects linked with an imperfection or problem in the source code were identified. Furthermore the meetings were used to consult with the expert on the data correctness and to also discuss and verify analysis results such as the alignment of defects to the correct releases. Hence these meetings facilitated in gathering accurate defect information and consequently increased precision of the quantitative analysis results. The meetings could be classified as participatory research activities. The results were documented using meeting notes.

[a] ISO/IEC/IEEE 24765 defines defects as "a problem which, if not corrected, could cause an application to either fail or to produce incorrect results", and "an imperfection or deficiency in a project component where that component does not meet its requirements or specifications and needs to be either repaired or replaced".

Analysis of the quantitative data was done through the aid of descriptive statistics. Descriptive statistics targets creation of an understanding and provides an overall description of the most important details of the data [17]. In this case it is used to explore significant characteristics of the defect data reported across releases.

2) Qualitative Data

As part of the qualitative analysis, interviews were conducted with employees that were involved with the product development before the transfer. The purpose of the interviews was to investigate subjective opinions on quality across different releases during software evolution. An overview of the interviewees, their roles and responsibilities, and the number of years of being involved with the product are given in Table 2. Due to convenience and availability for face-to-face interviews, only employees from Site-A were interviewed.

All interviewee are no longer involved with development of the product. The latest involvement was terminated in 2010. This was the System Architect and Technical Manager (interviewee 10 in Table 2) who was involved with the product before, during and after the transfer.

TABLE II. INTERVIEWEES ROLES AND RESPONSIBILITIES

No	Role (Before Transfer)	Responsibility	Involvement (No. of Years)
1	Technical Coordinator for Customization	Leadership of a team of developers in customer customization	**~ 9 years**
2	Designer of the maintenance activities 2008-2009	Design, development and testing	**~ 8 years**
3	Designer	Design and development	**~ 8 years**
4	Solution Architect	Design and communication of the product architecture to developers and the verification and validation team.	**~ 3 years**
5	Team Leader (10 people)	Design and development	**~ 7 years**
6	Application Integration Tester	Testing	**~ 9 years**
7	Tester	Non-Functional Testing	**~ 8 years**
8	Development Manager	Unit Management for around 20 persons. Responsibility for the product.	**~ 9 years**
9	Test Line Manager	Test Line Management	**~ 7 years**
10	System Architect and Technical Manager	Analysis of Architecture and source code source code architects	**~ 10 years**

Quality is multifaceted thus questions were formulated to ensure different angles and perspectives were covered during the interview process. This ideology is similar to that proposed in the McCall model [18]. In particular, maintainability, reliability and reusability were selected as the key source code quality characteristics as suggested in [19, 20], while questions pertaining to understandability, modularity and complexity were used to detect inconsistencies in the information provided by the interviewees. The approach of using such safeguards is similar to that suggested by Xenos and Christodoulakis [16].

All interviews were recorded with the consent of the interviewees. Responses were transcribed and coding was done placing responses into categories. Depending on the context of discussion modularity, complexity and understandability can be linked to reliability, maintainability or reusability aspects. Thus responses were placed into maintainability, reliability or reusability category depending on the theme, topic or context of interview discussion or response. This process of grouping qualitative data according to patterns and relation is similar to the Typological analysis method [21].

In addition to the interviews, archival data and records, such as documentation and information posted on the company's intranet webpages, were also reviewed to gain deeper insight into the product's evolution. After the interviews, email communication with the interview participants was also used to clarify any unclear details pertaining to the discussions from the interviews and evolution of the product.

IV. Results and Analysis

In this section we first discuss how the product evolved from its initial release in 2007 (see Section 4.1), and then present our findings regarding the quality of the product (see Section 4.2) and how different GSD settings influence the quality (see Section 4.3).

A. Evolution

Table III illustrates the product release history data for six major releases, including subsequent changes in releases since their announcement. The number of defects in the table is multiplied by an anonymous factor for confidentiality reasons. Analysis of the size of each release is important for several reasons. First of all it is fair to assume that the larger the release, the more potential defects it may contain. Secondly, the larger the release, the more time it may require to be delivered.

The findings suggest that the source code for the product's releases grew in size by approximately 56% between releases R1 and R6. The main sources of additional changes (and thus LOC) in subsequent releases were new features, customizations for specific customers and/or defect corrections for each release, which are assessed qualitatively in relation to the main release.

TABLE III. Release History

Release	Year	Release Size (in LOC)	Increase in LOC (relative)	Delivered after (in months)	Defects[b]
R1	2007	910 974	*Unavailable*		100
R2	2007	1 004 814	+5,7%	5	222
R3	2008	1 100 881	+19,1%	6	91
R4	2008	1 217 545	+19,1%	6	361
R5	2009	1 334 120	+0,2%	8	427
R6	2010	1 424 943	+15,0%	12	801

b. Number of defects is multiplied by an anonymous factor for confidentiality reasons

Column "Increase in LOC" in Table III denotes percentage increase in LOC for each release i.e. difference in LOC since introduction of the specific release on the market until end of 2011. This difference is due to customizations for specific customers or defect corrections only. For example since the availability of R2 on the market, R2 has increased in LOC by 5,7% due to defect corrections and customizations.

Against expectations the size of additional changes was not proportional to the number of defects found. This may be explained by refactoring efforts, which are not evident from the purely quantitative data analysis. Thus, explanations were further sought through interviews. Information obtained from the interviews revealed that refactoring was seldom performed before R4.

As one interviewee revealed measuring the lines of code will not tell the complete story, if the major refactoring work is done. He explained that new functionality was added without any refactoring efforts, and only after R4 the code was being revised.

This means that LOC is not a reliable measure of work effort. However, from the quality point of view

it still illustrates the size of the legacy code that is maintained and thus is of interest for our analysis.

The evolution of the product suggests linear growth in size between subsequent releases of approximately 9-10% between the first five releases and approximately 6% between R5 and R6. This is consistent with the first and sixth laws of evolution: law of continuing change [5] and law of continuing growth [5], respectively. With the implementation of mainly new features and defect corrections, each software release has been larger than the predecessor. Consequently, growth in sizes of source code artifacts may increase complexity of releases [1].

Some interviewees revealed that as the product increased in size over time, they had more difficulty isolating defective source code components.

Thus the increase in complexity in the course of evolution and was also taken into consideration during retrospective analysis of the software quality. For example, it could also explain why release frequency slowed down over time as the release cycle history indicates.

B. Evolution and Quality

Like many software organizations, defect data in Ericsson is used to evaluate the quality of the product development. Table III shows that there has been a gradual increase of defects after R3, with a significant increase in defects for R6.

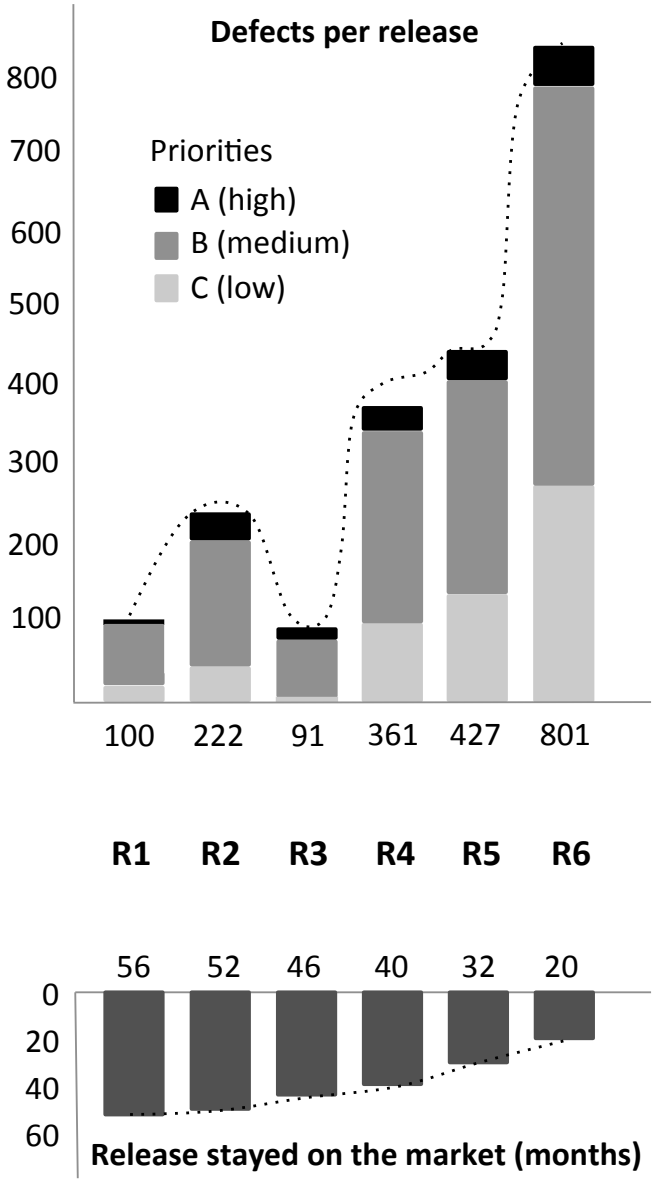
Seven of the interviewees related the increase in defects reported primarily to the increase in size of releases and the increase in complexity of features.

In order to understand the criticality of this trend the defect data was then broken down to different priority levels as shown in Figure 1. Priority "A" represents the highest priority level and "C" represents the lowest level, according to the significance or severity of the reported problem. Interestingly, while the amount of defects of "B" and "C" priorities is proportional and repeats the overall defect curve, there is no dramatic increase in the high priority problems since R3.

As mentioned earlier, the total number of defects per release has been collected not only during the actual development of the initial version of the release, but also during its subsequent maintenance activities. Figure 1 depicts the length of the lifetime of and the relative number of defects per release. For example, R1 has been on the market for 56 months, while R6 only for 20 months. We would expect to see more defects for older releases, since the likelihood of new or more defects emerging increases as the system usage increases [22].

Some interviewees pointed out that increase in the number of customers over time could have influenced the number of defects reported.

When studying the data, however, no linear dependency was observed. Thus we conjure that more powerful factors have determined the defect curves for example effectiveness of testing process.



c. Number of defects is multiplied by an anonymous factor for confidentiality reasons

Figure 1. Defects and Priorities per Release

Motivated by our findings a more detailed look on the defect data focusing on the chronological course of defect reports was created and discussed with the interviewees (see Figure 2). The data illustrates a sharp rise and decline in the number of defects shortly before and/or immediately after each release. Interviewees attributed this trend to the implementation of new features in each new release.

Interviewees revealed that, there would be often a peak in the number of reported defects after implementing new functionalities. However, eventually the number of defects reported would decline and stabilize.

Interestingly, R6 has several peaks that qualify as pre- and post-release increase in the number of defects, and these are significantly larger than for the previous releases. To investigate the reasons for this trend we further discuss the impact of a transfer on product quality.

C. *Software Transfer and Quality*

As noted earlier, the evolution of the transfer indicates two interesting trends – there is a growing amount of defects in R6 and the delivery of that release required much longer time. To illustrate these trends in the light of product evolution we show the distribution of defects reported from the beginning of 2007 until the end of 2011 as well as the transfer milestones between releases R1 and R6 in Figure 2. The actual transfer took place after R4 as shown in the figure. R5 was released during the transfer period whilst R6 was released after the transfer.

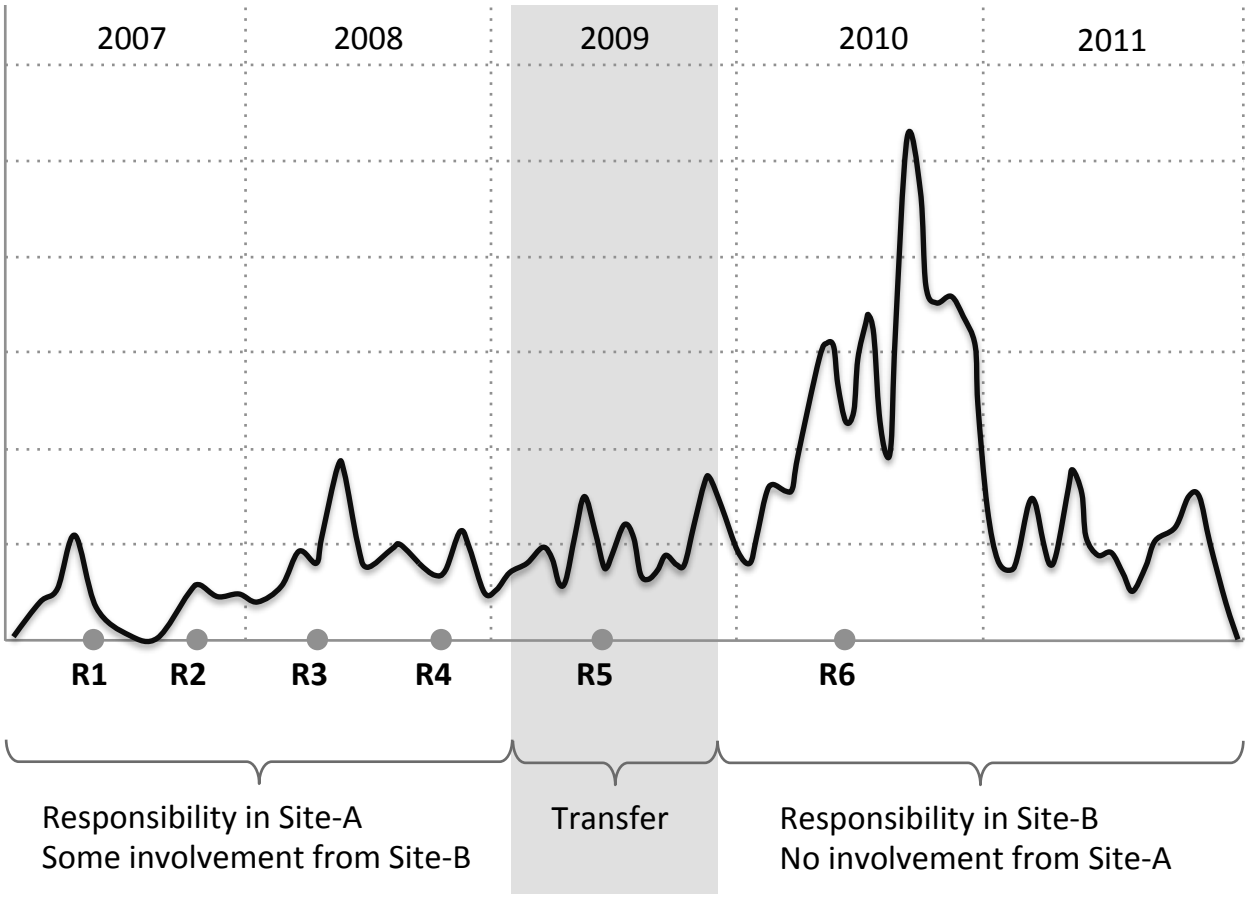


Figure 2. Defects Across Releases and Transfer Period

Though confounding factors, such as system usage increase, may have had a role in the exposure of defects, our observations suggest that the transfer of the product from Site-A to Site-B might have been the main cause of the significant increase When discussed with the interviewees, a common opinion was expressed.

Eight out of ten interviewees noted, that the average defect-rate across releases was rather stable, and that introduction of the new developers from Site-B into the development caused various challenges.

Employees from Site-B were temporarily moved to Site-A for competency and practical training in the preparation for the transfer. Involvement of the Site-B led to several observations of the learning process as discussed by the interviewees.

Some interviewees pointed out that the lack of prior engagement with the product meant that they were not aware of dependencies between source code artifacts or components, and thus of the ripple effect that certain changes had. However, as time passed the necessary knowledge and competences were built up and defects related to such cases decreased.

Two interviewees perceived cultural difference to have an important role in explaining why the defect curves changed.

The interviewees perceived that the new site had different ways of approaching quality goals.

Although the limited knowledge and experience has a profound influence on the quality, we conjure that implementation of different testing processes might have also contributed to the post-transfer increase in the number of defects reported.

V. DISCUSSION

In this paper we have studied an evolution of a product that was first developed in one location and then transferred to another location of the same company. We have explored how quality of the product changed throughout its lifetime on the basis of quantitative defect analysis and qualitative observations elicited from developers.

The hard facts indicate that the size of the product experienced a linear growth with each release, the delivery cycles became larger and the number of defects was uneven and notably increased for the last release. However, there are many other factors that may alter the interpretation of objective measures over different releases [25, 28] and further explanations were sought through interviewing product developers.

These additional observations suggest that as the size of the product grew, complexity increased and maintaining the legacy code became challenging. This confirms the second law of evolution – the law of increasing entropy/complexity [6].

Other factors discussed as potential sources of quality concerns were: the number of customers, exposure to the market, and complexity of the new features developed. However, the relative stability of the defect curves in the first five releases (with exception of the third release) suggests that the seventh law, the law of declining quality during product evolution [5, 7], is not confirmed. This means that it also cannot be used to explain the sudden significant increase in the last release of the product.

One of the emerging findings in this study relates to the impact of a software transfer on product quality. Previous research has established that a transfer is a nontrivial and demanding task, more so for large complex software products [18]. This study confirms related findings of an initial decrease in quality after a transfer [14, 18], and attributes it to a limited knowledge and experience with the product. While task familiarity is important for performance [24], a transfer disrupts the continuity of the knowledge. Some of the discussed consequences revealed problems with isolating defects and ripple effects caused by the failure to evaluate the impact of changes. It is worth noting that the decrease in the number of defects in 2011 in Figure 2 shall not be perceived as the quality improvement, as it simply illustrates the "tale" of the defect reports for R6. To understand whether the quality is improving or not, a latter release shall be inspected.

Transfers have been also blamed for initial decrease in efficiency [14, 18, 26]. For example, Mockus and Weiss [26] report that in their study, full recovery of productivity for development work after a transfer was approximately 15 months (excluding training period). Although we have not measured productivity in our study, we indirectly support the decrease in efficiency by observing the post-transfer increase in release cycles. While the first four releases were delivered on average twice a year, the transfer activities slowed down the delivery of the fifth release to eight months, and the delivery of the sixth release to a full year. Notably, 19 months passed since the last release by the time of

our study, and the seventh release is not ready yet. This suggests that companies transferring software products shall expect not only a quality decrease, but also significant effect on the release frequency.

Finally, perceptions of quality elicited from developers in our exploratory study were consistent with that obtained from the quantitative analysis of the defect data. Data obtained from the interviews provided invaluable explanations pertaining to certain variances in the defect data. We thus highlight the importance of identifying environmental variables that should be considered during defect data analysis.

VI. Validity Threats

At the time of this study, approximately three years had passed since the interviewees terminated their involvement with the product. Hence recollecting events or details pertaining to the course of events was problematic. However, relating the interview questions to major milestones during the evolution (for example, involvement of the new employees into the team, beginning of the transfer) helped the participants to remember certain important aspects. The list of interview questions also contained some questions that were used to check for inconsistencies in information provided. The idea of using such a set of questions is similar to the approach of using safeguard questions by Xenos and Christodoulakis in [12].

Observations gathered through the interviews represented only the perspective of employees at Site-A. This is a limitation of our study and may have biased the findings. Interviews with Site-B could have helped to identify the different factors that contributed to the increase in post-transfer defects. Nevertheless, the defect trend and distribution over-time was consistent with the viewpoints collected through the interviews conducted, thus we trust that our major conclusions are reliable.

Accuracy of the quantitative analysis results relies on the accuracy and precision of the defect data collection process. Furthermore, defects need to be appropriately linked to the actual releases e.g. a defect can be linked to a change or defect fix implemented in release R2 but only found in R4. Hence, to increase precision of results the test and verification expert at the case company helped with gathering defect data and tracing it to release history information. In addition, taking into account human error, defect data used in the quantitative analysis is treated as approximation rather than exact measures.

Quality is a multifaceted concept that can be described from different viewpoints and notion of quality differs between roles. This makes the selection of characteristics challenging in any quality study. We alleviated this issue by using a perspective on quality similar to that proposed by designing interview questions tackling different angles of software quality; an idea similar to that proposed in the model by McCall et al. [18].

VII. Conclusions and Future Work

In this exploratory study, quality analysis is conducted across releases of a software product that has been transferred between two sites of a company. The evolution of the product confirms the sustainable growth in size, while the quality levels varied between the releases and were subject for further investigation.

The analysis of the possible reasons for the changing quality revealed that the growth in size is related to the increase in complexity of maintaining the legacy code of the product. Complexity of the new features included in particular releases, as well as the number of customers and exposure to the market are also factors that have to be taken into account. However, the major impact in the evolution of the product studied according to our results was caused by the transfer of development and responsibility to people who had limited or no previous engagement with the product. Our findings suggest that companies that plan to transfer development from one site to another shall expect an initial decrease in quality and increase in release cycles. Although our findings are inconclusive about the recovery period, the fact that the work on the second post-transfer release is yet to be finalized indicates that the new site is still challenged by the work on the product.

Results from this study are limited to the context of the company, such as locations of the sites and complexity of the product. However, we believe that our conclusions regarding the implications of software transfers on quality shall be relevant for other software companies that transfer work on the global scale.

For future work, we plan to continue monitoring the defect reports and delivery cycles, as well as interviewing developers from the receiving site, in order to better understand the changes in ways of working and identify practices that can alleviate the recovery after a transfer. Additionally, we plan to differentiate between customer defect reports and internal testing reports in the future data analysis, and add the actual number of customers per release to get a better understanding of the market impact.

ACKNOWLEDGMENT

We thank Professor Claes Wohlin for his valuable feedback and discussions, and Ericsson employees for their help in retrieving and analyzing the data. Without their support this study would not have been possible. Ericsson Software Research and the Swedish Knowledge Foundation fund this work under the grants 2009/0249 and 2010/0311.

REFERENCES

[1] T. Mens and S. Demeyer, *Software evolution*. Springer, 2008.

[2] M. W. Godfrey and D. M. German, "The past, present, and future of software evolution," *Frontiers of Software Maintenance*, FoSM, 2008, pp.129-138.

[3] P. Bourque and R. Dupuis, "Guide to the Software Engineering Body of Knowledge 2004 Version," *Software Engineering Body of Knowledge 2004 SWEBOK, Guide to the*, 2004.

[4] Iso Iec, "Systems and software engineering -- Vocabulary," *ISO/IEC/IEEE 24765*, no. 1, 2010, pp. 1–418.

[5] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution - The Nineties View," in *Proceedings of the 4th International Symposium on Software Metrics*, Washington, DC, USA, 1997, p. 20–32.

[6] L. A. Belady and M. M. Lehman, "A model of large program development," *IBM Systems Journal*, vol. 15, 1976, pp. 225–252.

[7] M. M. Lehman, "Laws of Software Evolution Revisited," *Computing*, vol. 1149, 1996, pp. 1–11.

[8] C. F. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, Aug. 1999, pp. 493–509.

[9] D. Šmite and C. Wohlin, "Strategies Facilitating Software Product Transfers," *IEEE Software*, vol. 28, no. 5, Oct. 2011, pp. 60–66,.

[10] L. Chang-Peng, L. Bin-Shiang, L. Yen-Sung, and W. Feng-Jian, *A validation of software complexity metrics for object-oriented programs*, vol. vol.1. Hsinchu, Taiwan: Nat. Chiao Tung Univ, 1994.

[11] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, Jun. 1994, pp. 476–493.

[12] M. Xenos and D. Christodoulakis, "Measuring perceived software quality," *Information and Software Technology*, vol. 39, no. 6, 1997, pp. 417–424.

[13] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *27th International Conference on Software Engineering (ICSE) Proceedings*, 2005, pp. 284– 292.

[14] D. Šmite and C. Wohlin, "Lessons learned from transferring software products to India," *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[15] D. Smite, C. Wohlin, R. Feldt, and T. Gorschek, "Reporting Empirical Research in Global Software Engineering: A Classification Scheme," in *IEEE International Conference on Global Software Engineering (ICGSE),* 2008, pp. 173–181.

[16] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, Dec. 2008, pp. 131–164.

[17] C. Robson, *Real World Research 3e*, 3rd ed. John Wiley & Sons, 2011.

[18] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality," vol. I, Nov. 1977.

[19] International Organization For Standardization Iso, "Software engineering - Product quality - Part 1: Quality model," vol. 2, no. 1, 2001, pp. 1–25.

[20] C. W. Krueger, "Software reuse," *ACM Comput. Surv.*, vol. 24, no. 2, Jun. 1992, pp. 131–183.

[21] L. M. Given, *The SAGE Encyclopedia of Qualitative Research Methods, vol 1 and 2*. Sage Publications, 2008.

[22] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, Sep. 1980, pp. 1060– 1076.

[23] D. Šmite and C. Wohlin, "Software Product Transfers: Lessons Learned from a Case Study," in the *5th IEEE International Conference on Global Software Engineering (ICGSE)*, 2010, pp. 97–105.

[24] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development," *Organization Science*, vol. 18, no. 4, 2007, pp. 613–630.

[25] M. M. Lehman, D. E. Perry, and J. F. Ramil, "Implications of evolution metrics on software maintenance," in *International Conference on Software Maintenance Proceedings*, 1998, pp. 208–217

[26] A. Mockus and D. M. Weiss, "Globalization by Chunking: A Quantitative Approach," *IEEE Software*, vol. 18, no. 2, 2001, pp. 30–37.

[27] D. N. Wilson and T. Hall, "Perceptions of software quality: a pilot study," *Software Quality Journal*, vol. 7, no. 1, 1998, pp. 67–75.

[28] J. Li, N. B. Moe, and T. Dybå, "Transition from a plan-driven process to Scrum: a longitudinal case study on software quality," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, New York, NY, USA, 2010, pp. 13:1–13:10.