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

Ronald Jabangwe, Darja Šmite

International Conference on Global Software Engineering (ICGSE)

2012

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 teams from different locations get involved in the development of a software product during its evolution. However, empirical evidence that demonstrates the effect of changing 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 Ericsson, a large multinational corporation, 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: Global Software Development; Software transfer; Software evolution; Software quality; Empirical case study

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 eight laws of software evolution between 1974 and 1985 [2–4].

Global software development (GSD) projects, in which development activities are carried out in various settings such as offshore and distributed development, are increasingly a prevailing phenomenon. Thus it is not uncommon that software evolves in a number of GSD settings. Consequently 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, for example, [5] and [6]. 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 [7]. An example of a classification of different settings can be found in the work of Šmite and Wohlin [8]. In our case, the company studied has been involved in a number of scenarios in which development work was transferred to offshore sites for further development or maintenance. 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 [8]. We thus study the effect of a transfer as a complication in the course of software evolution and evaluate software quality across releases.

In their work Belady and Lehman [2] postulate the bases and peculiarities associated with software evolution for large systems built for commercial use. The laws describe the inevitability associated with evolving systems. Due to market pressure and demands the product used in this study is subject to change in characteristics and increase in size with, for example, the addition of new features, which the laws of evolution link with implications on complexity and quality. Thus the following laws are specifically of interest in our study. The first law of evolution, the law of continuing change, and the sixth law of evolution, the law of continuing growth, suggest the importance of implementing and adapting changes over time so as to sustain or increase user satisfaction [2–4]. Software evolution is a phenomenon that arises as a result of these progressive changes. The second law, the law of increasing entropy/complexity, and the seventh law, the law of declining quality, hypothesize the link between software evolution and the potential decline

in quality [2–4]. Our work is inspired by the laws of evolution and driven by the curiosity to better understand these implications in the context of software transfers.

This paper describes 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 before, during and after a transfer. The product used in this study was first developed in one location, then jointly with an offshore site and then had the development work transferred to the offshore site. We propose that transferring development work during software evolution to a site that has little or no past experience with a product may have implications on quality as the new site build up competency and climb the learning curve.

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 offers 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 discussed in Section VI. The paper ends with the conclusions 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. [9] 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 [10]). Complexity was also studied subjectively through perceptions of complexity expressed by developers. Similarly, Xenos and Christodoulakis [11] 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 [12]. Maintenance and evolution as terms are often used interchangeably in academia and industry [1], [13]. The Software Engineering Body Of Knowledge (SWEBOK) [14] 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 Belady and Lehman [2] more studies have been carried out on the relation of software evolution and quality. 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) [15][16]. Results from the study conducted by Nagappan et al. [16] 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 [15]. 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 [8]. 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 [17].

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 [18].

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 [19]. 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 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 [17] (in the article the transfer for this particular product is referred to as Project B).

Šmite et al. [7] 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 [8] and the classification in [7].

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	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
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 [19]. 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 [20]. 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 software 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. Therefore we use a slight modification of the two definitions from the International standard ISO/IEC/IEEE 24765[a] [12]. 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 increased the precision of the quantitative analysis results. The results were documented using meeting notes.

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 [20]. In this case it is used to explore significant characteristics of the defect data reported across releases.

2) Qualitative Data

As a 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.

None of the interviewees are involved with development of the product any longer. The latest involvement was terminated in 2010 as seen in Table 2. It is worth noting that several interviewees were involved with the product before, during and after the transfer. This is important, because we explore what happened in the post transfer period.

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 [21]. In particular, maintainability, reliability and reusability were selected as the key source code quality characteristics as suggested in [22] [23], while questions pertaining to understandability, modularity

[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".

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 [11].

TABLE II. INTERVIEWEES ROLES AND RESPONSIBILITIES

No	*Role*	*Responsibility*	*Involvement (No. of Years)*
1	Coordinator and Techincal Lead	Leadership of a team of developers involved in product customizations	~ 9 years
2	Developer	Design, development and testing in the maintenance activities (2008-2009)	~ 8 years
3	Developer	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	Leadership of a team of appr. 10 developers involved in design and development	~ 7 years
6	Tester	Application integration testing	~ 9 years
7	Tester	Non-functional testing	~ 8 years
8	Development Manager	Unit Management for appr. 20 people. Responsibility for the product.	~ 9 years
9	Test Line Manager	Test Line Management	~ 7 years
10	System Architect and Technical Manager	Analysis of the architecture and source code architects	~ 10 years

All interviews were recorded with the consent of the interviewees. Coding was conducted by placing interview quotes into categories of specific quality factors. Depending on the context of discussion modularity, complexity and understandability were linked to reliability, maintainability or reusability aspects. This process of grouping qualitative data according to patterns and relation is similar to the Typological analysis method [24]. Table III contains the number of interview quotes that were found to be associated with three quality attributes and confounding factors that were noted by the interview participants as having potential influence on defect occurrence. Some interview quotes counted in the analysis were not mutually exclusive. For example, certain quotes for reliability were also found to be either associated with maintainability or reusability or both aspects of the system. Hence some quotes were counted in multiple categories.

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 evolution. After the interviews, email communication with the interview participants was also used to clarify any unclear details.

TABLE III. INTERVIEW QUOTES

Categories		*No. of quotations*
Quality Attributes	Reliability	48
	Maintainability	59
	Reusability	24
Confounding factors on defect occurrences	Differences in culture or ways of working	8
	Change in product attributes (i.e. Increase in size or complexity)	17
	Lack of prior engagement	10

IV. RESULTS AND ANALYSIS

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

A. Evolution

Table IV 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 IV. RELEASE HISTORY

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

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

Column "Increase in LOC" denotes percentage increase for each release i.e. difference in LOC since

introduction of the specific release on the market until end of 2011. 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 explained, 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 [4]: law of continuing change and law of continuing growth, respectively. With the implementation of mainly new features and defect corrections, each software release has been larger than the predecessor. Thus, 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 needs to be taken into consideration. For example, it could 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. Table IV 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 [25].

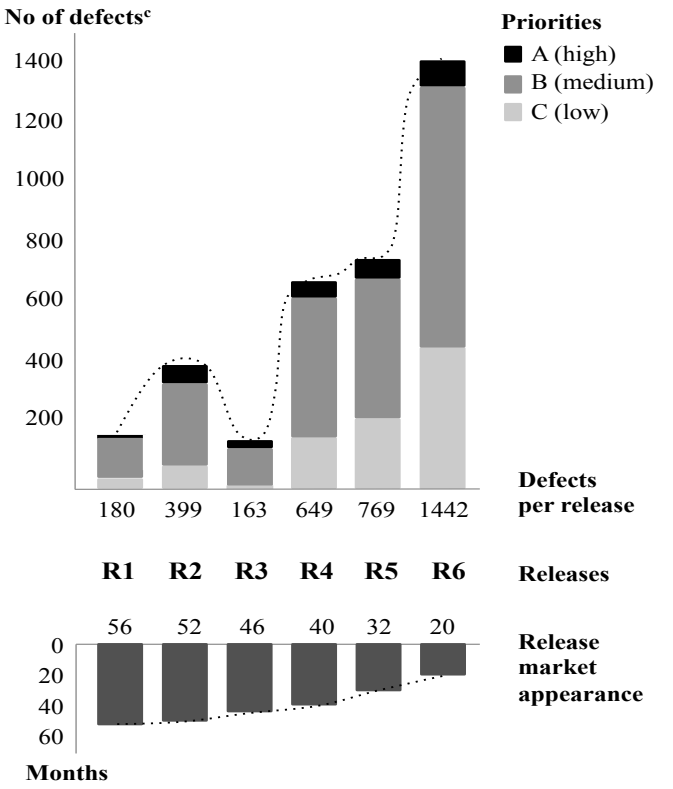
Some interviewees pointed out that increase in the number of customers over time could have influenced the number of defects reported.

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.

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).

We noticed that there is 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.



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

Figure 1. Defects and Priorities per Release

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.

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 when transferring the knowledge necessary to build up competency and aligning quality views

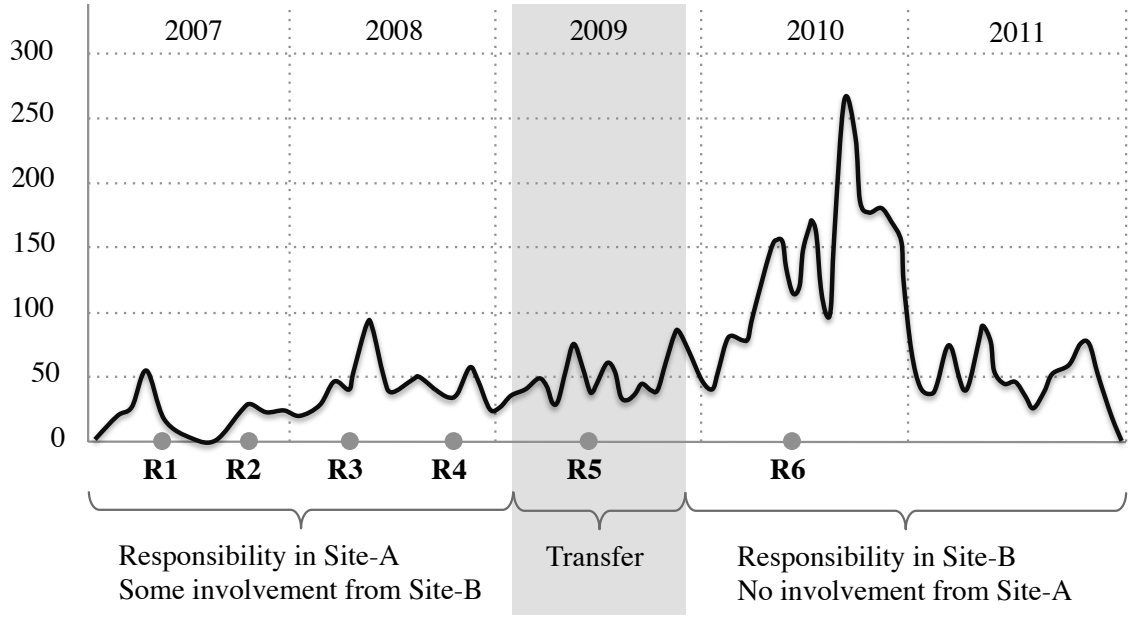


Figure 2. Defects Across Releases and Transfer Period

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.

According to the results from the interviews there was a misalignment of the notion of quality between employees from Site A and those from Site B. As a result the ways of meeting quality goals and ways of working differed between employees from the two sites. This potentially influenced defect occurrence during the involvement of employees from Site B.

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 [26] 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 [4].

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 [4], 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 [17]. This study confirms related findings of an initial decrease in quality after a transfer [17][27], and attributes it to a limited knowledge and experience with the product. While task familiarity is important for performance [28], 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 [17][27][29]. For example, Mockus and Weiss [29] 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 the length of 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, but the seventh

release has not seen the light. This suggests that companies transferring software products shall expect not only a quality decrease, but also significant effect on the release frequency or scope.

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 and not relying on solely quantitative analysis of quality through defects.

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 [11].

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 formulating interview questions designed to tackle different angles of software quality; an idea similar to that proposed in the model by McCall et al. [21].

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. This is a logical consequence, since the growing complexity of the legacy code during software evolution puts much more pressure and demands on the developers, and thus changes in project staffing have more dramatic effects. 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 the length of 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 and productivity shall be relevant for other software companies that transfer work both on the global scale and locally. In fact, we conjure that transfers from one building to a neighboring building may have same consequences.

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 Dr. Kai Petersen 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.

REFERENCE

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

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

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

[4] 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.

[5] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality", In 30th International Conference on Software Engineering. ICSE 2008, p. 521.

[6] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? An empirical case study of Windows Vista," 2009, pp. 518–528.

[7] 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.

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

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

[10] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng*., vol. 20, no. 6, pp. 476–493, Jun. 1994.

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

[12] *Systems and software engineering -- Vocabulary*, ISO Standard no. 1, pp. 1–418, 2010.

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

[14] P. Bourque and R. Dupuis, *Guide to the Software Engineering Body of Knowledge*, Software Engineering Body of Knowledge , 2004.

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

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

[17] D. Šmite and C. Wohlin, "Lessons learned from transferring software products to India," *Journal of Software Maintenance and Evolution: Research and Practice*, Published online 27 July 2011

[18] J. Li, N. B. Moe, and T. Dyb\aa, "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 2010. New York, NY, USA, 2010, pp. 13:1–13:10.

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

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

[21] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality. Concepts and Definitions of Software Quality," Nov. 1977. vol 1.

[22] *Software engineering - Product quality - Part 1: Quality model*, ISO Standard vol. 2, no. 1, pp. 1–25, 2001.

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

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

[25] M. M. Lehman, "Programs, Life Cycles and the Laws of Software Evolution," *Proceedings of the IEEE,* vol. 68, no. 9, pp. 1060–1076, 1980.

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

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

[28] 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, pp. 613–630, 2007.

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