A Case Study of a Very Large Organization

by

Colin Mark Werner

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2011

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Very Large Organization (VLO) is an organization that produces hardware and software, which together form products. VLO granted access to data pertaining to seven different products and their development projects. One particular product is of interest to VLO since it was not as successful as the other products. The focus of this thesis is to study the problematic product and compare it to the other six products in order to draw some conclusions regarding the problematic product. The goal of this study is to indicate areas of improvement, which can help VLO improve future products.

This thesis explores and answers the following research questions focused around the problematic product. Was the product indeed a failure? If so, what caused the product to fail? What indications that the product would fail were evident during the product's development? What could VLO have done in order to prevent the product from becoming a failure? What can VLO learn from the failure? Are there data from the non-problematic products that indicate what VLO excels at?

This thesis analyzes the data from all seven products and their projects in order to answer the research questions. Analyzing the non-problematic products is important in order to draw comparisons to the problematic product. As a result of this research, this thesis uncovers a variety of issues with the problematic product and identifies six areas for possible improvement. These six areas are: hardware research and development, decoupling of software from hardware, requirements management, maximal use of resources, developer order and priority of vital features, and schedule alignment. This thesis concludes that even though none of these six problematic areas can be pinpointed as the *singular* root cause of the problematic product's failure, addressing these problems will improve the likelihood of product success.

Acknowledgements

First, I would like to thank and acknowledge Daniel Berry, my supervisor, for his continued and excellent support throughout graduate school and finally with the completion of this thesis.

I would also like to thank my two reviewers, Joanne Atlee and Derek Rayside, for providing their input into this thesis. I would also like to thank my parents for their support throughout my graduate career and this thesis.

Finally, a big thank you to Erin. She provided me with all the support one could ever ask for. I thank her for her understanding throughout my graduate career and especially with writing this thesis.

Thank you all!

Table of Contents

List of Figures

List of Tables

Chapter 1
INTRODUCTION

The practice of software engineering is a difficult subject to research, requiring detailed examination of actual software product development projects.[19] One of the largest problems is that commercial software products are typically very large and extremely costly to develop, and no company will develop one for the sole purpose of concluding research about its development. However, on the flip side, decreasing the size of the product to make it more affordable as a research subject will have the ill effect of losing external validity, since the development of a smaller scale product is not typical in a commercial setting[2]. To begin, a small experiment should be first performed with a high degree of control in order to provide internal validity. To obtain external validity, a case study of the development of a large-scale product must be performed, which is where a research institution can benefit from partnering with a commercial organization that produces software. Moreover, it is very likely that the commercial organization can also benefit from the results of partnering with the research institution. This thesis is an example of Very Large Organization (VLO) partnering with a research institution.

VLO is a technological firm that produces hardware and software for customers around the world. This thesis provides an in-depth analysis of seven products, Pd1, Pd2, Pd3, Pd4, Pd5, Pd6, and Pd7, produced by VLO. Each product has its own development project, Pj1, Pj2, Pj3, Pj4, Pj5, Pj6, and Pj7, each of which provided the data to be analyzed. A lot of data from VLO were analyzed to examine the process, the products, the projects, and the organization itself. There is one product, Pd6, and its accompanying project, Pj6, that garnered elevated interest from customers and employees. In particular, Pd6 was deemed to be a failure. Therefore, the six other products were analyzed in order to compare and draw some conclusions with respect to Pd6. This thesis poses a set of research questions that will also be answered through analysis of the seven projects developed by VLO.

This thesis is organized as follows: Chapter 2 has a detailed description of the products and the associated research questions that this thesis will answer. Chapter 3 details how and what data were gathered. Chapter 4 analyzes, compares, and discusses the extracted data for each project. Chapter 5 provides a variety of possible solutions to some of the problems raised in Chapter 4. Chapter 6 describes related work and some other potential areas of future work. Chapter 7 draws some conclusions about the research performed. Finally, Appendix A contains all of the extracted data.

Chapter 2
DETAILED DESCRIPTION OF PRODUCTS AND RESEARCH QUESTIONS

This section outlines the products produced by VLO, introduces a series of research questions, and discusses other important information that may impact the answers to the research questions. VLO produces hardware-and-software products, each of which includes an embedded systems layer and an applications layer. Generally speaking, when VLO releases a new product, the product consists of new hardware accompanied with its own new embedded system and an applications layer. Occasionally, VLO may release new software for hardware that is already in market, thus having both old and new hardware running the same software. The software release process model used for the majority of releases is based on an iterative waterfall model in which each subsequent release builds directly on top of the previous release. Using an iterative model does not mean that software production is strictly iterative. There exist multiple exceptions in which new features or additions are included in a product in order for the product to succeed in a highly competitive market.

2.1 Product Components and Life Cycle

In this thesis, seven different products at VLO are analyzed; each of these products has a corresponding development project. The seven products together took VLO more than three years to develop and release. Each product consists of three components: hardware, an embedded system, and an applications layer. Each component is part of a layered architecture, in which each layer depends on the lower layer. See Figure 1 below.

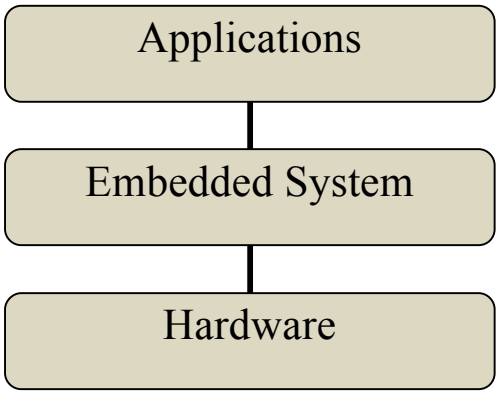

Figure 1: Layered Architecture

While it is possible to work concurrently on the three components of the system, it is neither ideal nor easy. Typically, the hardware and the embedded system are highly cohesive and require a more complex integration. For instance, complications tend to arise with the introduction of new hardware that operates in a manner slightly different from the manner that old hardware does. These complications are hard to predict and generally cannot be discovered until the actual hardware is delivered. Thus, parallel work on the hardware and the embedded system is difficult. Unfortunately, more problems exist at the highest level of abstraction, in the application layer, which depends directly on the embedded system to interact with the hardware. However, work on the application layer can still be carried out concurrently with the implementation of a non-functioning application programming interface (API), also known as a stub interface, which corresponds to what the embedded system will eventually supply. Any change in hardware can be masked behind the façade of the embedded system so that the development of the application layer can proceed. Problems arise when an entirely new component is being added to a product, and the addition involves work across all three layers. Unfortunately, this situation occurs quite often in the fast-paced and high-demand technology market. However, this difficulty provides more incentive to develop a general solution that can be applied under many different scenarios.

VLO does not invest a lot of time or energy on requirements engineering. Generally speaking, a team of marketing specialists initiates a product release. The first task for a marketing specialist is determining what requests are in the market, i.e., what the customers desire. Typically, these desires are new features on top of a previously released product. The requests from the marketing specialists are formally captured in a document, which also notes the priority of the request and is then passed on to a team of technical experts and executives. The team of experts and executives evaluate the requests and negotiate with the marketing specialists until a succinct list of approved requests is drawn up. Generally speaking, only the highest priority requests are accepted. Other lower priority requests are typically deferred until a later release. The list of approved requests acts as the requirements document and is presented to the development teams for design and implementation.

2.2 Research Questions

For this thesis, the seven products analyzed are Pd1, Pd2, Pd3, Pd4, Pd5, Pd6, and Pd7. Each product has a corresponding development project denoted by Pj1, Pj2, Pj3, Pj4, Pj5, Pj6, and Pj7. The product of particular interest is Pd6, which is viewed as largely unsuccessful, both internally at VLO and externally by customers. Although the internal view of Pd6 is not publically expressed, there is nothing to stop a customer from exercising the right to free speech and publishing his or her opinions. Based on these

published opinions, it is clear that Pd6 is widely acknowledged as at least a partial, if not a complete, failure. This thesis analyzes the data from the seven projects with the ultimate goal of drawing some conclusions about the development of Pd6. It should be noted that the majority of the information gathered is from the associated development project of the applications portion of each product. In order to fully analyze Pj6, this thesis will answer the following five research questions (RQs). RQ1: What caused Pd6 to fail? (Answered in Section 4.2) RQ2: What indications were there during Pd6's development that Pd6 would fail? (Answered in Section 4.2) RQ3: What could VLO have done in order to prevent Pd6 from becoming a failure? (Answered in Chapter 5) RQ4: What can VLO learn from the failure? (Answered in Chapter 5) RQ5: Are there indications from Pd1, Pd2, Pd3, Pd4, Pd5, or Pd7 that indicate areas that VLO is excelling at? (Answered in Section 4.1.7).

2.3 Product Classification

Before analysis can be performed and comparisons are made, it is important to classify each product according to a variety of criteria. First, the relative size and complexity of each product is comparable; each successive product builds upon the previous. Second, each product is classified whether it is being released on in-market hardware, new hardware, or possibly both. Furthermore, it is important to note how many different hardware platforms a particular product supports.

Product	In-Market	New Hardware	Number of Hardware Platforms
Pd1	Yes	Yes	7
Pd2	No	Yes	4
Pd3	Yes	No	12
Pd4	No	Yes	2
Pd5	No	Yes	4
Pd6	No	Yes	2
Pd7	No	Yes	1

Table 1: Product Classifications

The classifications of the seven products, shown in Table 1, are used to evaluate whether a valid comparison can be made between two products. These classifications are important for two reasons. The

first is that not all of the products may be usefully compared with others. The second is that these classifications may help to explain any oddities in the data. However, the number of hardware platforms shown in Table 1 is slightly misleading and deserves further clarification. If the hardware is new, the way in which it is new should be documented. For example, if the hardware is changing from a 800 MHz processor to a 900 MHz processor produced by the same company with the same specifications, then it can be reasonably assumed that even though the hardware is new, the amount of work needed would be less than that for adding a dual-core 2.2 GHz processor or adding an entirely new component such as a touch screen to a personal computer.

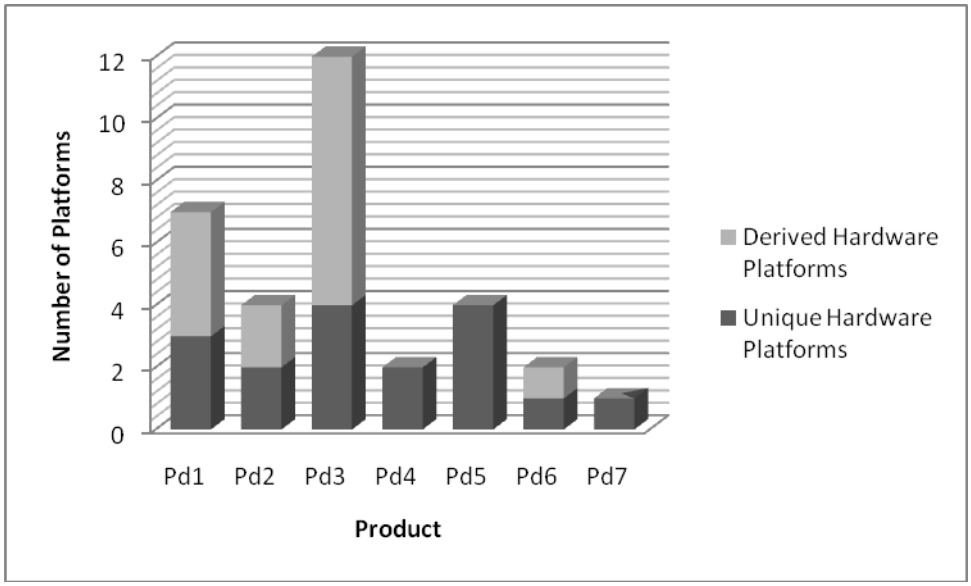

Figure 2: Breakdown of Hardware Platforms

The graph in Figure 2 displays a more detailed breakdown of hardware platforms. Here again, more explanation is required. First, each unique hardware platform has a physical design, a set of components, and a layout that is completely distinct from any other. However, taking any one platform as a base platform and subsequently changing or adding small components on top of the base platform creates a derived hardware platform. The number of unique and derived hardware platforms is useful information for the analysis in Chapter 4.

2.4 Organizational Behaviour

It is important to document how VLO was evolving throughout the seven projects. VLO grew a considerable amount between Pj1 and Pj7. The growth was both in depth, as multiple satellite offices were opened, and in breadth, as additional employees were hired at existing locations. This thesis does

not discuss whether the growth experienced by VLO was sustainable. However, the organizational behavior is an important consideration as it can have a profound effect on the employees and therefore on the products. It is also important to note the market in which VLO is a major competitor has become even more competitive as other organizations entered the market with new similar products. The extra competition induced more customer demands on VLO to produce high quality competitive products. Thus, a major shift in organizational behaviour occurred to deal with the added demand. Since the competition affected all the competing organizations similarly, it is not considered in this thesis.

Chapter 3
DATA MINING METHOD

This section describes the methods taken to acquire the data studied in this thesis. However, since these projects all occurred in the past, research can be performed only with saved data and through discussions with current employees that were employed during a particular project. Part of the problem with data mining is finding where to look for data, especially relevant data. The information provided by VLO was extremely beneficial for analysis in this thesis and for the software research community, even though only a small amount of information was available to study. In particular, seven consecutive projects, each to develop a single product, were selected and data about the development project for each product were collected from source control repositories, change request (CR) reports, customers' acceptances, project schedules, and employee surveys. The majority of the data collected pertains to only the application layers for the products. These data are acceptable because the application layer is the highest level of abstraction and the last to be completed. Also, the application layer is the last to start, so there is a smaller window of opportunity to discover problems, whereas the other layers have more time to uncover and fix potential problems. The application layer is also more interesting to study because of its heavy dependency on lower level components. Finally, some of the projects do not contain a complete set of data; in particular, each of Pj1 and Pj2 is missing the CR reports.

3.1 Source Control Repositories

The most important information gathered is the number of source commits (SCs) from the source control repository for each project. A SC is a single source commit by a single developer to the source control repository. The data is simply counted as each SC occurred, so there is no advance notice of outstanding changes to be committed. A SC may consist of a list of requested items, each of which is a feature request, a CR, a software bug fix, or a non-functional change such as an optimization or a change to the document. Therefore, the actions a developer may take for a single SC can be any combination of editing an existing file, deleting an existing file, or adding a new file to the source control repository. It is also important to note that this thesis does not classify the SCs based on content; for simplicity, each and every SC is considered to be of equal value.

The information gathered with each SC consists of the SC number, which is a sequential number assigned by the source control repository, the date the source code associated with SC was committed to

the source control repository, the name of the developer who completed the SC, and a brief description of the SC items written by the developer. However, all of this information is superfluous for the purposes of this thesis. Therefore, this thesis counts only the number of SCs per month by project. These compressed data, along with some empirical analysis can be found in Appendix A. For the purposes of this thesis, a project is code complete when the last SC has been submitted to the source control repository. Therefore, the code complete percentage at any given time is comparing the running total number of SCs at that time to the final total number of SCs, which is when the last SC is committed. This definition is useful because the projects discussed are all past projects, and therefore the actual code completion dates are known for each project. In general, a product is released to customers before its project is code complete and therefore provides interesting points to analyze. In particular, a useful statistic for comparison is the percentage of code completion of a project when the first acceptance is received from one of the customers of its product.

3.2 Change Request Reports

At VLO, a process exists that must be followed when a new requirement or a change in an existing requirement is desired, usually due to a customer's influence. When a change is required, a CR must be submitted. A CR includes details of what is desired, why the change is required, and by when the change is expected. Generally speaking, a CR blocks a project release; otherwise the importance of the CR is minimized. Once a CR has been submitted, it is assigned a unique number in the CR-tracking system at VLO. The marketing team is the first to see the CR to decide whether its requested change is worth pursuing. Next, the teams that would be involved in implementing the change must provide an impact analysis of the change. This analysis includes estimates for development, testing, localization, certification, or any other areas that may be affected by the change. Normally, it is very difficult to compile a complete list of all the teams that may be affected by a particular change. Therefore, many a CR does not include a complete list of estimates. This deficiency is itself a problem, as other work that was not in the original impact analysis is often discovered and consequently delays the completion of a CR. After the information is gathered, the CR is sent to a CR committee, which decides whether to approve or reject the CR.

If a CR is approved for implementation, a new CR for each affected component is created in the CR-tracking system. Each new CR contains all the information documented in the original CR and includes a reference to the original CR number. Each new CR is sent to the appropriate team for design and implementation. In a perfect scenario, no other work is necessary and each team is able to complete its

work within the estimated time frame. Occasionally, complications arise because not all the appropriate teams were consulted prior to approving the original CR. These complications prevent VLO from completing a CR in a timely fashion.

For this thesis, the following CR data were obtained: CR number, title, project, submitter, and date submitted. More information from the CR database is available; however, this information lies outside the scope of this thesis. Also, only approved CRs were queried, as rejected CRs had no impact on the project. However, Chapter 6 explores how the discarded data may be of significant use in future research. Also, the number of CRs per month by project is compiled and can be found in Appendix A.

3.3 Customers' Acceptances

There are many customers worldwide that use products from VLO. However, note that there is a distinct difference between a customer of VLO and a consumer of the products produced by VLO. In particular, VLO sells its products to customers; each customer then resells the product to the consumer. VLO does not sell directly to the end users. Thus, it is an important part of each customer's reputation to only sell products that will appease customers. As an organization, it is vitally important for VLO to consider the customers first, as they pay VLO for the various products. Therefore, satisfying the customers is the most important priority for VLO. For each product, the database contains which customer accepted, when it was accepted, and a variety of other details pertaining to the various hardware platforms and software versions.

3.4 Project Dates

Project dates play a very important role in answering the RQs. Unfortunately, VLO does not have an easily accessible database with dates. Therefore, it is very difficult to find out what the first estimated shipping date was for each project, as well as any subsequent changes to a particular date or why a change in the data occurred. However, an actual date of acceptance is easy to obtain, as it is generally a much-celebrated day. Schedule slippage is a very good indication of a problematic project. VLO does not have a specific location or database with the various dates. The dates were compiled through a variety of sources, including e-mails, project documentation, and directly from employees.

3.5 Employee Opinions

Perhaps the most valuable resource at VLO is its thousands of employees. For the purposes of this thesis, various employees were asked for their opinions. The resulting discussions proved to be useful because

they indicated which projects were problematic. These opinions provided data that were not officially documented by VLO. Even though any particular employee may be biased, the opinions of several employees help determine and validate answers to the RQs.

3.6 Document Database

VLO maintains an internal document database utilized by the entire organization, including the software division. Theoretically, all documents written within the company are housed in the database, however, in practice a document is often not entered into the database, or a document once entered is not kept up to date. Each of the seven projects had a unique placeholder within the database, so all related documents were extracted from the database and analyzed. The types of documents found included meeting minutes from status, stakeholder, and executive meetings. However, not all projects had all three different types of meetings documented in the database. Since these meetings were generally held on a weekly basis, the minutes were a valuable asset in gathering a snapshot at any given time during the project. Typically, the minutes noted the current set of target dates, noting any slippage or foreseeable slippage, future and completed milestones, and finally any risk items that were of concern. The documents offer a snapshot perspective of a particular project at a single point in the past. The documents explicitly reflect a series of chronological events that are an integral part in the development of a particular project. This information is essential when trying to piece together exactly what occurred during development and provides insights that may have been forgotten. All of this information is useful when answering the RQs.

Chapter 4
ANALYSIS OF DATA

This section begins with a brief discussion of Pj1, Pj2, Pj3, Pj4, Pj5, and Pj7. A more detailed analysis of Pj6 follows, which draws on the data and analysis of the other six non-problematic projects. In Table 2, below, each project is listed with the summarized data from Appendix A and a relative rank for each datum. The data for each project include the project's numbers of SCs, CRs, and acceptances; the project's development time measured in months; the code completion percentage at the apex of the project's bell curve of SCs over time; and the code completion percentage at the project's first acceptance by any customer. At the bottom of the table, the maximum, minimum, and average of each datum is calculated. The code completion percentage for a particular project at first acceptance is simply the percentage of SCs which have source code committed to the source control repository at the date of the project's first customer acceptance. The code completion percentage at the apex of the bell curve column indicates what percentage of SCs had source code committed to the source control repository at the peak of the project's bell curve on the number of SCs versus time graph. More detailed graphs for individual projects can be found in Appendix A.

Project	Number of SCs	Rank	Number of CRs	Rank	Number of Acceptances	Rank	Time	Rank	Percentage Code Complete at Apex of Bell Curve	Rank	Percentage Code Complete at First Acceptance	Rank
Pj1	5184	6	85	5	702	4	26	2	58.06	2	72.45	4
Pj2	8508	3	62	6	253	6	18	6	38.72	6	**88.15**	**1**
Pj3	7185	5	94	4	**2090**	**1**	**30**	**1**	43.81	3	87.92	2
Pj4	12767	2	**215**	**1**	1152	3	25	3	42.45	5	72.05	5
Pj5	7956	4	105	3	1184	2	22	5	**34.72**	**7**	**65.79**	**7**
Pj6	**17262**	**1**	148	2	309	5	23	4	42.55	4	67.88	6
Pj7	**4477**	**7**	**49**	**7**	**46**	**7**	**15**	**7**	**68.24**	**1**	87.47	3
Maximum	17262		215		2090		30		68.24		88.15	
Minimum	4477		49		46		15		34.72		65.79	
Average	9048		108		819		23		46.93		77.39	

Table 2: Comparison of Data from the Seven Projects

4.1 Non-Problematic Projects

This section explores the six non-problematic projects: Pj1, Pj2, Pj3, Pj4, Pj5, and Pj7. First, the data for each project is discussed in order to answer the RQ5, "are there indications from Pd1, Pd2, Pd3, Pd4, Pd5, or Pd7 that indicate areas that VLO is excelling at?". Pd1, Pd2, Pd3, Pd4, Pd5, and Pd7 were not perfect, however, the general consensus, internally and externally, is that each of these products was certainly not a failure. It is important to note that Pd1 is not the first product produced by VLO. In fact, VLO had already earned a reputation of producing first-class products before the inception of Pj1.

4.1.1 Pj1

When VLO began working on Pj1, the organization was fairly mature, thus even though Pj1 is the first project analyzed, the performance of Pj1 is not a measure of the maturity of VLO. However, this project was one of two projects that involved releasing software to hardware platforms that already existed in the market, which is referred to as an in-market release. This project spanned three unique hardware platforms and an additional four other variants for a total of seven different hardware platforms, which ranks second only to Pd3 for total number of hardware platforms. Unfortunately, this project lacked any meeting minutes or schedules from the document database, but the database did contain some documents that outlined some of the lessons learned and complaints from the perspective of the developers for this project.

The resonating view from within the organization was the large number of hardware platforms for this single project. The employees also felt that this project had too many CRs to handle in a single project. In general, a large number of CRs would normally indicate poor requirements engineering. However, the data indicate that Pj1 had only eighty-five CRs, which ranks Pj1 fifth most for total number of CRs among the seven projects. The fact that employees voiced an opinion about the number of CRs and that Pj1 only ranks fifth most with respect to number of CRs is alarming. This indicates that VLO did not listen to the employees, especially since the number of CRs almost triples for Pj4.

Pj1 ranked second lowest in the total number of SCs, which is relatively low among the projects. The graph of the number of SCs versus time for this project shows its bell curve peaking during the seventh month of development. This curve is typical also for most of the projects; therefore it is reasonable to compare the curves of the various projects. In general, a project starts slowly, but as it proceeds, the amount of work gradually increases until the apex of the project's bell curve is reached and then it gradually tapers off until completion. At the apex of the bell curve, 58.06% of all SCs for Pj1 had source

code committed to the source control repository, which is slightly higher than the average among the seven projects.

A more interesting statistic is that Pd1 received its first acceptance when the code was 72.45% complete, which is slightly below average for other projects. This is not necessarily a negative sign, since it is common for one hardware platform to have priority over another and thus to be worked on prior to other hardware platforms. Pj1 was being released in-market as well as with new hardware, so priority was given to the new hardware over the in-market hardware. These circumstances explain the slightly lower than average percentage at which the first acceptance was received compared with the other six projects. Another interesting statistic about Pj1 is that it ranked in the bottom half or below average in every category, except for time taken to develop. This project was the second longest in time to develop, possibly indicating a correlation between the number of hardware platforms and time. However, numerous other mitigating factors can cause an increase in the duration of the development of a project.

Finally, all of these data indicate that Pj1 does not compare well with Pj6. This is mainly due to the high number of supported hardware platforms. Also, the fact that Pj1 was developed for in-market devices inhibits comparisons with Pj6.

4.1.2 Pj2

Pd2 involved two unique and two derived hardware platforms for a total of four hardware platforms. Pj2, to build Pd2, experienced an accelerated start, culminating with only 38.72% of the SCs being submitted by the apex of the bell curve at the three-month mark. However, this project stretched out for an additional fifteen months past the apex of the bell curve. The number of SCs for Pj2 ranks third most, even though it is still below the average. Pj2 had fewer CRs than Pj1. However, the marketing team stated that there were a large number of CRs deferred from Pj1 to Pj2 that were now being deferred to Pj3, so more and more CRs were being deferred. The increased number of deferred CRs is a direct side affect of the fact that fewer changes were being accepted. Also, the increased number of deferred CRs indicates that the organization is not simply accepting CRs blindly and including them in projects and that the change control process at VLO is working. However, in both Pj1 and Pj2, employees expressed a concern with the number of CRs, so even with the decreased number of total CRs, there is a recurring problem with CRs and likely indicates an area that VLO could further improve on.

Pj2 received the first acceptance with 88.15% of SCs having source code committed to the source control repository, which is the highest percentage amongst the seven projects. This high percentage

indicates that Pj2 had a full set of dedicated resources for this project, which was duly noted in the project minutes. Also, given that this project finished in eighteen months, which is the second shortest, indicates that all the resources necessary were available and allocated to work on Pj2. This percentage is also a clear indication that customers did not prematurely accept the product. However, such an indication is valid only in hindsight since the project is now complete and the total number of SCs for this project will remain static. Analyzing and using this statistic as a real-time indicator of when a project is complete and can actually be released is discussed in Chapter 6.

The high number of SCs, the low number of CRs, the high percentage of SCs committed by the first acceptance, the short time frame to receive the first acceptance, and the relatively short time frame from project inception to completion indicate that Pj2 is a model project for VLO. Ideally, the process and practices used for this project should be implemented and used on future projects.

4.1.3 Pj3

Pj3 is unique among the six non-problematic projects. Pd3 did not include any new hardware platforms; it was comprised solely of an in-market upgrade to the software of virtually all previously released hardware platforms that met the minimum requirements for this project. This project had four unique hardware platforms and another eight variants for a total of twelve hardware platforms, which is the most of any of the seven projects. However, even though Pd3 had the highest number of hardware platforms, Pj3 ranks only fifth most for the total number of SCs. This relatively low number of SCs indicates that the number of new features included in this project was limited and that the purpose of this project was to bring the vast majority of in-market products to the same level of software and to fix any major problems that customers had previously reported. The number of CRs for Pj3 is comparable to those for Pj1 and Pj2, although Pj3 has still slightly more. Unfortunately, the initial concerns for this project are not as well documented as Pj1 and Pj2; thus it is impossible to determine whether the number of CRs or the number of hardware platforms was still continuing to be a major concern to employees at VLO.

Pj3, with a duration of thirty months, took by far the longest to develop. Pj3 also had, by far, the most acceptances, almost doubling those of the second ranked project. It should be noted that each acceptance is for one hardware platform. Since there were twelve hardware platforms supported by this product, the most of any of the products, it is reasonable to expect that there would be more acceptances for this project than any other. That Pd1 and Pd3 had the highest number of hardware platforms and that Pj1 and Pj3 had the longest project development time suggest that a project's development time increases with the number of hardware platforms of its product. However, there are other factors that may increase a

project's development time, for example the number of CRs can cause a project duration to be extended. Although the official schedule dates were not documented, the various status reports indicate that Pj3 suffered very little schedule slippage, which is impressive given the size of this project, although not surprising since this product involved no new hardware platforms.

Most notably, Pj3 had the second highest percentage of SCs submitted by the first acceptance with 87.92%. This is a clear indication that the software was not prematurely accepted; although as previously noted this indication is useful only after the project is complete. Also, it is important to note that this project did not release any new hardware platforms; therefore the hardware platforms were mature enough and well enough known by VLO. Thus, the level of quality expected and achieved on this product was higher than normal. Unfortunately, this project has the least in common with the other projects, and thus makes for a difficult comparison, especially to Pj6, which is the project of interest.

4.1.4 Pj4

Pj4 was the first project for a product that was released for only new hardware platforms; also Pd5, Pd6, and Pd7 were released for only new hardware platforms. This similarity should allow for easier comparisons among Pj4, Pj5, Pj6, and Pj7. Also, given that this project was released at a date closer to the release date of Pj6, the results of such comparisons are more meaningful. Pj4 consisted of creating two entirely new hardware platforms. It had the second most changes and more notably the most CRs of any project. These last two factors indicate that there was a great amount of work required in order to support the two new hardware platforms, which contained components that were considerably different from those of any previous hardware platforms. Even though each of the hardware platforms contained components that had never been included into a project before, these components were evolutionarily different, not revolutionarily different. Thus, even though there was a great deal of work involved, this sustainable growth was incremental and similar to growth that VLO had experienced before and therefore was more than capable of. Finally, this project can easily be compared with Pj6 due to the similarities across all data; the ease of comparison will be vital in answering the RQs.

It is also noteworthy to consider that the first acceptance for Pj4 was received with only 72.05% of the total SCs for this project, approximately another 3,500 SCs were committed after the first acceptance. This well-below average number is alarming because it indicates that the project was prematurely accepted. The number of hardware platforms supported by this project cannot justify this below-average statistic since there were only two hardware platforms. However, as previously discussed, the priority of one of the hardware platforms could have been higher than the other, and this was the case for Pj4. The

first acceptance received for Pj4 was for the higher priority hardware platform, or the lead platform, as this is confirmed in the data. The lower priority hardware platform did not receive an acceptance until month nine, which was two months after the first acceptance for the lead hardware platform. When the lower priority hardware platform received the first acceptance, the code was 82.42% complete, which is above average for the seven projects and 10% above the higher priority platform. In the end, no schedule slippage was documented for this project, and the minutes indicate that acceptances were even received from customers before the originally planned date.

Pj4 ranks third highest for total acceptances and had the third longest development time. However, Pj4 did not experience a typical bell curve with respect to SCs versus time. Pj4 started with a high volume of SCs for the first five months and then the volume steadily decreased over the following ten months. The reasons for this peculiar curve are unknown but are also not very concerning. Overall, this project has the most in common with Pj6 and thus can be more easily compared with Pj6.

4.1.5 Pj5

Pd5 was originally slated to have three hardware platforms, each of which was new and unique to VLO. However, during month twelve, VLO added to Pd5 an additional fourth hardware platform, which, like the other three, was completely new and not derived. The knowledge of the late addition is instrumental in understanding the analysis of this particular project and is essential for comparing Pj5 with Pj6. Amazingly, even with the unplanned additional hardware platform, the originally scheduled dates slipped by only two weeks.

Overall, the data for this project are average when compared to the other six projects. This project ranks fourth in total number of SCs, third in number of CRs, and fifth in total development time. Despite the fact that the number of SCs was average, this project is one of only two, the other being Pj6, which experienced a secondary spike in the graph plotting the number of SCs versus time. However, the late addition of the fourth hardware platform is clearly a reasonable explanation for this second spike.

This project is not average with respect to the percentage completion in the bell curve of the number of SCs versus time, for which it ranks last with 34.72%. Although, this project does have two distinct spikes, the data are slightly skewed because of the second spike, which is attributed to the late addition of a hardware platform. Also, Pj5 ranks last with only 65.79% of SCs at the time of the first acceptance, which notably occurred before the fourth hardware platform was even added to this project. Therefore, these statistics for Pj5 are skewed and not as meaningful when compared to that of the other projects.

Overall, the data indicate that this project was extremely different from the other six. However, the addition of the new hardware platform is an explicit reason for this difference. Yet even with the additional hardware platform, this project did not appear to be problematic because it did not cause delays for the originally planned hardware platforms. In fact, if the data for Pj5 resembled those for the other projects, then Pj5 would actually be considered problematic because the late addition of the hardware platform would have caused problems for the first three hardware platforms. In the end, even though the data from both Pj5 and Pj6 appear to be similar on the surface, the data in Pj5 can be explained by the late additional hardware, thus differentiating Pj5 from Pj6. Therefore, the two projects cannot be easily compared.

4.1.6 Pj7

Pj7 is by far the smallest project, which explains why this project ranks last in the total number of SCs, total number of CRs, total number of customer acceptances, and development time. Also, this product featured a single hardware platform. The data for Pj7 indicate that VLO can succeed with incremental advancements on previous projects. The scheduled dates for this project were met with few exceptions. It is also a positive note that Pj7 occurred after Pj6, and thus VLO had recovered from Pj6. Pj7 does indicate that VLO has an excellent understanding of the technology it produces and is able to do what VLO knows quite well. The data from this project simply indicate that VLO knows the details of this particular technology niche and is able to accurately predict how to move forward in an evolutionary manner. However, when revolutionary changes occur, VLO does not succeed in the same manner, as indicated by the data for Pj6.

4.1.7 Indications of Successful Outcome

This section summarizes the answer to RQ5, which questions, "are there indications from Pd1, Pd2, Pd3, Pd4, Pd5, or Pd7 that indicate areas that VLO is excelling at?". In fact, the data for Pj1, Pj2, Pj3, Pj4, Pj5, and Pj7 indicate that VLO does many things well. First, the data indicate that VLO has refined the process to incrementally release products over time, which is good practice according to Ruhe et al and "implicitly reduces many of the risks associated with delivering large software projects".[10] Even though the new products are not revolutionary, they are nonetheless advancing forward with market demand. The data for these six projects also demonstrate that VLO can take a previous product, upgrade the hardware, add some novel changes to the software, and launch a new product. The data clearly show that

VLO is able to accurately estimate the effort required to implement new features, because VLO is an expert in this domain. Thus, VLO knows its niche very well.

4.2 The Problematic Project: Pj6

Pj6 is the project with an elevated interest for a variety of reasons. The primary reason is that consumers believed Pd6 was a failure, or at least not the booming success that VLO had intended. It is very difficult to determine when a product is indeed a failure. Nevertheless, in the eyes of the customers, Pd6 was not ready for launch when it was released, and it did not meet customers' expectations, especially when compared to competitors' offerings. It is also important to note that the customers were eagerly anticipating the launch of Pd6 with high expectations caused by their delight with the previous products from VLO and the promises made by VLO about Pd6. The expectations for Pd6 were enormous, which VLO failed to meet. Thus, Pd6 is considered to be a failure by customers. As described by Charette[5], software failures can be detrimental to an organization so it is vital for VLO to avoid having recurring failures.

Moreover, the view of Pj6 from employees at VLO is not overly positive. Many employees thought the project deadlines were too tight given the requirements for the product. It is important to note that there was an extremely hard deadline for this project that VLO had committed to with the various customers. As such, many employees believed the project was launched too soon, despite the fact that employees worked exceedingly hard in order to complete the requirements specification for the product. The employees' view is an excellent indication that Pd6 was a failure, and it should serve as a clear lesson to the management at VLO that the employees know a product and the state of its project well and may offer some insight as to the readiness of a project for release.

In order to fully analyze Pj6, comparisons are made to other projects in order to pinpoint which factors contributed to the outcome of Pj6. First, the projects should be compared at a high level in order to determine which components of the projects are similar. In Section 4.1, each of the six non-problematic projects is discussed in detail with respect to: hardware platform types, development time, in-market releases, various data points from Appendix A, and other peculiar details about each project. This analysis determines whether useful comparisons can be made to Pj6.

To start, Pj6 must be summarized. Pd6 consisted of one new unique hardware platform and one additional derived platform, which had one small component of hardware changed, which is a relatively low number of hardware platforms compared to other products with new platforms. In terms of hardware

platforms, Pj6 most resembles Pj4 and Pj7. However, comparisons can be made also with Pj2 and Pj5. The rest of this section explores other aspects of Pj6 that may provide clear answers to some of the RQs posed in this thesis.

4.2.1 Revolutionarily New Hardware Component

The hardware platform designed for Pd6 was revolutionary because VLO was attempting to design and include a hardware component that had yet to be developed. This hardware platform was not evolutionary as were the hardware platforms of Pd1, Pd2, Pd3, Pd4, Pd5, and Pd7. The revolutionarily new hardware component was deemed necessary by VLO in order to maintain an important part of VLO's reputation. However, the introduction of the revolutionarily new hardware component was also very risky because it had yet to be developed. Even under these circumstances, the potential benefit outweighed the risk because the revolutionarily new hardware component would differentiate the product amongst the competition. Therefore the decision was made to move forward with the design and creation of the revolutionarily new hardware component to be included in Pd6.

When creating a product with a revolutionarily new hardware component, an organization must perform adequate research prior to committing the revolutionarily new hardware component to the project. The amount of research that VLO undertook before the decision was made to include the revolutionarily new hardware component in Pd6 is unknown; however, there is evidence which indicates not enough prior research was performed. However, there does not exist a strict enough criterion in order to conclude whether enough research was performed; just the outcome of the project indicates that the technology was not thoroughly researched. This topic is discussed in more detail in Chapter 6.

Problems with the hardware platform emerged right from the beginning in Pj6. The first version of the hardware platform, which was a simplified version of the expected final hardware, was delayed. The implications of hardware platform delays are discussed in Section 4.2.3. Despite the delay, VLO proceeded with the regular development process with the initial version of the hardware platform. It is standard industry practice to have hardware platform revisions throughout the development period of a project. However, the differences between versions are generally small tweaks, as most of the major changes should have been performed during the research and development of the prototype and first version of the hardware platform. With Pj6, the differences between hardware revisions were critical changes to the fundamental functionality of the hardware platform. These sorts of changes had many implications and a particularly colossal impact on the software, especially the embedded system that controlled the hardware platform and ultimately impacting the application layer. The initial hardware

platform version was received with the caveat that major changes were expected. This initial version of hardware was so drastically different than the planned second version that much of the software development was postponed until the second version of hardware had been received. The third version of hardware was received shortly after the second, with only minor tweaks from the second.

Unfortunately, by month four, less than five months prior to the planned launch date, the third version of the hardware was not functioning properly, causing problems that could not be solved in the software. Therefore, a fourth version of the hardware was necessary. In fact, the fourth version of hardware was actually a slight modification of the first version. The fact that the third version of the hardware was deemed inadequate shortly upon arrival indicates that not enough research was conducted during its research and development before committing it to the project. Nonetheless, a CR was initiated that required a fourth version and the appropriate process was followed. However, the CR was not officially approved until the end of month five, which was more than half way into the development time allocated for Pj6. Moreover, at this point, the project was already two months behind schedule.

When the fourth and final version of hardware was delivered, Pj6 was now into the eighth month, three months behind schedule. Finally, VLO could now proceed with completing the software, specifically the portion of the embedded system that controls the revolutionarily new hardware component of the hardware platform. However, it is important to realize that the first hardware version received at month three was practically identical to the fourth and final hardware version. In hindsight, the net minor changes to the hardware platform between the first and fourth version cost VLO approximately five months of delay. The five months that it took VLO to produce final hardware from the first version should have occurred during the research and development stage prior to committing the revolutionarily new hardware component to the project. In the end, evidence indicates that VLO had prematurely committed to the revolutionarily new hardware component before the specification for the hardware was complete. Once the second and third versions arrived, albeit late, the software teams could now begin implementation. However, the fourth version, which was essentially the same as the first caused the software teams to discard much of the software developed for the second version in order to develop the software nearly from scratch for the fourth and final hardware version. It is evident that not enough prior research was performed before committing the revolutionarily new hardware component to the project. The impacts that hardware changes have on software are discussed in Section 4.2.4.

4.2.2 Adding New Hardware Components

While Pd6 included a revolutionarily new hardware component, it also had another an evolutionary hardware component that affected hardware, the embedded system, and the application layer as much as the revolutionarily new hardware component. In fact, this evolutionary hardware component had been around for at least ten years and had been in competing products for the past couple years. In theory, this addition should not have been problematic for VLO since VLO has always been adding new evolutionary components to products. In actuality, the additional evolutionary hardware component caused problems in each of the three departments responsible for each layer in the architecture. These problems, compounded with the problems relating to the revolutionarily new hardware component, heavily contributed to the slow down of the project.

The additional evolutionary hardware component affected the embedded system and also included a very complex implementation at the application layer. This feature was heavily dependent upon the hardware, but it was dependent also upon software that was developed externally to VLO, complicating matters even more. Employees surveyed felt that this additional evolutionary hardware component was considered to be one of the major contributors to the delays extending the implementation of the application layer. The possibility of such a delay is an unfortunate aspect of developing a product with multiple layers. Any delay that occurs at a lower level usually occurs at the early stages of a project and does not cause an organization to panic. However, any delay at the upper layer, which is the last layer to be completed, is scrutinized by the entire organization because it is the last stage of the project to be completed. Ultimately, the team responsible for the application layer develops a negative reputation by the rest of the organization, even if the application layer is completed within the originally estimated time, but was late to start because it was dependent upon lower layers.

There are two questions related to this section that will be answered in Chapter 5. The first is whether an evolutionary hardware component should even be included in a project that also includes a revolutionarily new hardware component. The second is whether the dependency issues that the application layer experienced could have been avoided.

4.2.3 Hardware Schedule Delays

The initial version of hardware for Pd6 was delayed by eighteen days. Although this delay is relatively small, it occurred with the first version of the hardware, which could potentially delay future work. A

delay that occurs early on in a project is often impossible to make up, because an early delay could potentially be compounded and ultimately cause worse delays throughout the project launch date.

Relative to the original schedule, the second version of hardware was twelve days late. So the work leading to the second version actually made up six days. Unfortunately, thereafter the delay started to grow. The third version of hardware was twenty days late. So, the accumulated hardware delay increased by eight days. In the end, the final version of hardware was delayed by over sixty days, which is a large proportion of the originally planned schedule. The majority of the delays are attributed to the multiple hardware revisions described in Section 4.2.1. These data further reinforce the conclusion from Section 4.2.1 that not enough time was spent on prototyping the hardware platform before committing it to the product.

4.2.4 Implications of Hardware on Software

At VLO, as depicted in Figure 1, each product has three layers: hardware, embedded system, and application layer. The hardware and embedded system are intimately linked, especially since the embedded system is tweaked and optimized for the specific hardware platform used in the project. Nonetheless, an embedded system *can* be designed and developed to accommodate a variety of hardware platforms at the cost of non-optimal performance on all platforms. Of course, it is debatable whether this flexibility across hardware platforms is desirable; this is a decision that must be made by VLO. The products designed by VLO are part of a larger complete end-to-end system, including every hardware platform deployed. Therefore, the desire for software developed by VLO to support multiple hardware platforms is not a requirement since VLO also designs and manufactures custom proprietary hardware. Also, in order to maintain a competitive advantage, it is not desirable to allow competitors to be able to design and build hardware that is capable of running the software that VLO develops. Thus, it is reasonable that the embedded system would be heavily dependent on the hardware platform. However, given that VLO has released multiple product lines, it is expected of VLO to have a refined process to mitigate issues between software and hardware. This is especially important, as VLO should be able to predict the effect hardware has on software and perhaps even take a proactive step in developing an embedded system that is robust enough to remove as much of the dependency on hardware as possible.

At the top of the architecture hierarchy is the application layer. The application layer depends on the embedded system. It is conceivably easier to design an application layer that depends less on the embedded system using one or more of the industry-standard software engineering design patterns. Design patterns, such as the façade or the proxy pattern, could be used to create a defined layer of

abstraction between the application and embedded system. The extra layer of abstraction allows development of the application layer to be performed through a simulator. However, the application layer still needs to be tested in a production environment. Software and hardware dependencies are not unique to VLO. But, it is important for an organization to realize and anticipate delays at a lower level that may cause further delay at higher levels of abstraction.

4.2.5 Creation of New User Interface Design Team

While VLO was developing Pd6, the organization established a new team with the directive to research, design, and document all user interface requirements for all future projects. The first project that would employ the expertise of the new user interface design team was Pj6. The theory behind the new team was that if a single team of experts collectively developed the user interface requirements, then all aspects of the user interface would be consistent across all products. The expectations of this team of experts was to produce a user interface that was thoroughly thought out, researched, and tested through user experiments, user feedback, and focus groups.

The theory behind the user interface design team was excellent; however, as the team to began to work, three major complications arose with the team and thus falling prey to Brooks Law. First, the team was not working in parallel with other teams, who were dependent upon the user interface requirements to complete initial development estimates. The disconnect between the teams delayed other tasks such as test design and testing, which depended upon the user interface requirements. Second, the user interface team was not working in alignment with the project deadlines. Third, the user interface specifications produced by the user interface design team were not meeting the expectations set by management for Pd6. Despite the fact that development teams were not initially impressed with the user interface requirements, the implementation of the user interface proceeded. It was not until the implementation was practically completed that users across the organization started to discover that the user interface offered a poor experience. The user interface design team then rewrote the user interface specifications and the development teams implemented them with the same unsatisfactory results. The user interface design team kept coming back with more requirements, until finally during month six of development, upper management abandoned the entire user interface specification created by the user interface design team and dictated exactly how the user interface would appear.

In the end, the user interface specifications were not aligned with the project schedule and the results were unsatisfactory. The user interface design team also frustrated development teams that had worked on previous projects at VLO and had an excellent idea of how the user interface should look and feel.

Initially, the development teams disagreed with the user interface design teams, but the development teams still implemented the user interface specifications because of the expertise of the user interface team. Since the user interface is what the customer directly interacts with, it is imperative that the user interface be superb. Unfortunately, the user interface design team was new to VLO and designed the entire Pd6 user interface from scratch and thus became a contributing factor in the poor outcome of Pj6. The new user interface design team is another area that VLO should have spent more time researching before committing to a project, much like the revolutionarily new hardware component. The user interface design team had multiple complications through its infancy, but was able to recover from this project and to eventually produce satisfactory results in later projects.

4.2.6 Indications of a Failed Outcome

This section finishes answering RQ1, "what caused Pd6 to fail?", and RQ2, "what indications were there during Pd6's development that Pd6 would fail?". As Charette[5] described, it is very rare for a project to one or two reasons; however, some reasons may have a greater influence. Thus while answering RQ1 and RQ2, this section discusses the originally planned and actual dates for Pj6, number of SCs for Pj6, and internal opinions from employees. These three factors provide clear indications of failure. However, indications of failure do not necessarily imply that Pd6 is a failure.

First, let's examine the originally planned schedule and the actual dates. There were four important milestones tracked in Pj6. The four critical milestones are the hardware delivery, the completion of the embedded system, the completion of the application, and the first customer acceptance. The first milestone is hardware delivery. As discussed in Section 4.2.3, hardware versions were continually delayed, culminating with the fourth and last version being eight weeks late. Interestingly, the actual date of final hardware delivery was four weeks after the originally planned date for the first customer acceptance was to have been received by VLO, which was obviously delayed as well. The severity of the delay in the delivery of the hardware was the first indication that Pd6 was a failure.

The second and third milestones are related to the embedded system and applications layer software. As discussed in Section 4.2.4, software is dependent upon hardware. The embedded system feature completion date was delayed ten weeks and the application layer completion date was delayed fifteen weeks. Therefore, the application layer accounted for only five weeks of the total delay, however, with knowledge of the ten-week delay for the embedded system, the application layer could be perceived as the entire fifteen weeks. This longer perceived delay is an unfortunate aspect of being dependent upon other layers, but having no dependents.

The fourth milestone to examine is the date of the first customer acceptance. The first customer acceptance was received eleven weeks later than originally planned. However, it is also important to note VLO was under immense pressure to release this product by a specific date, which was set and agreed upon by both customers and VLO. Therefore, the customer may have prematurely accepted the product by a specific date, even though under normal circumstances the customer would not have accepted the product.

It is important to place the dates into context and to consider the relationship between the dates before judging whether they are indicators of failure. The development process, from start until the first customer acceptance, lasted sixty weeks. Therefore, the hardware delay consisted of 13% of the development schedule, the embedded system delay was 10%, and the application layer delay was 8%. Note that development continued after the originally planned sixty-week schedule on multiple major maintenance releases that continued for an additional forty-eight weeks, almost doubling the initial estimated development schedule. Even though maintenance releases are part of the development process, a maintenance release usually corrects catastrophic errors and the effort required does not usually double the development. There are three possible conclusions. The first is that the development estimate was inaccurate. The second is that other factors and dependencies caused the development process to be delayed. The third is or a mixture of both the first and second. The root cause is difficult to determine. However, many of the development delays revolved around the revolutionarily new hardware component and the difficulties with the addition of the new user interface design team. Ultimately, it took an additional 48 weeks to achieve the quality that VLO sought, even though the product was first released 48 weeks prior to achieve the desired product quality.

An important note about the development time is that it does not accurately reflect the planning process that VLO had undertaken prior to the beginning of the development process. In this thesis, the start of the development time of a project is the time stamp on the earliest piece of data discovered about the project. The planning stage started much before the nominal start date drawn from the data.

The SCs for a project are an indicator of the actual amount of work performed on a particular project. Pj6 has the largest number of SCs out of any of the seven projects discussed in this thesis. Also, as seen in the graph of Figure 13, Pj6 did not exhibit a one-peak bell curve like many other projects. Pj6's graph had two distinct peaks. A simple explanation for the two peaks would have been the addition of new features in subsequent releases, as seen in Pj7 or the addition of hardware as in Pj5. However, no new features or hardware were added, and therefore, the only explanation for the two peaks is that the project

was not ready at the actual launch and required more work. When the first customer acceptance was received, only 67.88% of all SCs for this project had been completed. The remaining 32.22% were completed in subsequent maintenance releases. The mere fact that Pj6's first customer acceptance occurred with so few completed SCs indicates that Pj6 continued to evolve and to mature for at least another year.

Finally, internal employee users of Pd6 across the organization were not impressed with the quality of Pd6 prior its to launch, especially in comparison to other competing products. Employees felt that the product was incomplete and rushed to market. The employees also thought the project had too many new complicated features implemented in a very tight schedule, which was marred by hardware complications and churn of user interface requirements. Employees thought also that Pd6 was released a year too soon, since development continued for an additional year until Pd6's quality had increased to the initially desired and expected level.

Other indicators of failure include multiple dates being delayed by a large margin, the relatively large number of SCs after the initial launch, and the plain and simple perception of employees at VLO. Notwithstanding these indications that the outcome was a failure, it is difficult to consider the outcome of Pj6 a failure because the product did indeed launch, even if it continually improved throughout the following year. The outcome of project was not considered a groundbreaking success, but the outcome of Pj6 cannot be considered an absolute failure since other organizations have failed to even launch products. In summary, and to answer RQ1 and RQ2, the indications of a failure can clearly be seen through major schedule delays and the mere fact that the product was failing to entice employees at VLO. Fortunately, there are numerous possible solutions to some of the difficulties faced by VLO. These are discussed in Chapter 5.

Chapter 5
POSSIBLE SOLUTIONS

As good as any organization might be, there is no organization, VLO included, which cannot gain from some form of improvement. The indications of a failed outcome discussed in Section 4.2.6 are the most obvious areas to start looking for possible solutions. Also, it is important for VLO to document each change made to the organization and to determine whether it had a positive, negative, or negligible impact in order to decide whether or not the change was worth implementing. Even though there are clearly defined indications that the outcome was a failure, it is very difficult to provide a viable solution for VLO. For example, the schedule delay is an obvious fault, but determining a solution is difficult because the root cause of the delay is unknown. Even though root cause may not be identifiable, it is better to at least make some improvements, even if they are small and do not actually solve the root cause. A small improvement may actually help uncover the root cause as well. The solutions in this section will answer RQ3, "what could VLO have done in order to prevent Pd6 from becoming a failure?", and RQ4, "what can VLO learn from the failure?". This section discusses possible solutions for the following six problem areas: hardware research and development, requirements management, maximizing available resources, maintaining focused releases, and alignment of schedules.

5.1 Hardware Research and Development

As discussed in Section 4.2.1, VLO encountered many problems with the hardware platform in Pd6, especially with the revolutionarily new hardware component. These problems highlight the importance of the prototyping stage. Also, since hardware is the basis for the entire project, adequate time must be spent on developing the hardware platform before committing it to a particular project. Even though VLO has produced hardware with a revolutionarily new hardware component in the past, all of the recently released products, except Pd6, were based on a steadily evolving hardware platform. Although it is not easy to determine the exact amount of research and development that must be performed in order for a product to succeed, there are clear indications when not enough research and development was performed. There has been considerable research performed on this aspect which VLO should take advantage of, such as Hooper et al,[21] which offers a "methodology and accompanying tools to aid users in identifying requirements before building a system".[21] Prior research has also been performed that compares on the effectiveness of various types of prototyping.[22] Competing organizations are all under immense pressure to produce the next best product, so it is all too easy for an organization to prematurely

release a revolutionary product in order to reach the market first. However, a premature release may not be in the best interest of the organization in the long term. As DeMarco described, that all late projects have one thing in common, they all started late.[4] Thus the hardware prototyping was started too late and ultimately caused the entire project to be late.

There exists an alternative to attempting to develop a revolutionarily new hardware component along with other new software features. The alternative is to invest an entire project schedule on the revolutionarily new hardware to ensure that the hardware is right and then simply take the software from the preceding product. This ensures that the single focus of the product is to introduce revolutionarily new hardware. It also provides an easy test harness as the new product can simply be compared to the old product, since all the software features should be the same.

The first indication that not enough research and development was performed at the beginning of Pj6 is that the first hardware version received by Pj6's software development teams was drastically different from the expected final hardware version, as the final hardware version was still being actively developed. In the end, VLO reverted to the simpler initial hardware version for product launch, but only after five months of development time had been wasted. The solution is to perform adequate research before project schedules are committed. It is also important to keep the rest of the hardware platform as simple as possible, so that adequate attention is focused on the revolutionarily new component. The prioritization of hardware components is discussed in Section 5.5.

5.2 Decoupling of Hardware and Software

Due to the delays that occurred with the design of the hardware platforms caused by the revolutionarily new hardware component and the addition of an evolutionarily new hardware component as discussed in Sections 4.2.1 and 4.2.2, it is a worthwhile investment for VLO to find ways to decouple hardware and software development. There are quantitative methods described by Kang et al[3] that could be applied to in order to refactor both software and hardware. The decoupling of hardware and software development would ultimately allow for parallel development of the hardware platform and the software allowing schedules to overlap, and ultimately leading to earlier detection of problems. Also, there are a multitude of other benefits to decoupling hardware and software, such as increasing effectiveness of debugging and increasing the possibility of reproduction.[24] Even if decoupling hardware and software development does not solve the root cause of project delay, the decoupling could lead to a more successful outcome if VLO undertakes another challenging product like Pd6 in the near future. Also, as previously discussed, the

decoupling of hardware and software development may cost VLO market share, as it could open a potential security hole as other organizations may be able to develop hardware that can run the software produced by VLO.

Even though decoupling hardware and software development may seem to be a great idea, decoupling also poses a technical question with serious ramifications that VLO must answer before considering implementing any decoupling: can VLO effectively decouple hardware and software development while maintaining control over what devices the software will operate on? VLO must carefully approach this problem. Ideally, a solution should be designed from the bottom up with appropriate amounts of preparation and requirements documents to be reviewed by teams across the organization. Although, decoupling hardware and software development is not easy, it would help alleviate the impact that hardware development delays have on software development, as was the case with Pj6. The most extreme decoupling involves the organization to develop only hardware or only software at any give time, this is also discussed in Section 5.1.

5.3 Requirements Management

Effective and efficient requirements management allows an organization to have a solid grasp on what features a product will support before the product is even developed. However, incorrect requirements management can cause a project to fail. Blyth et al indicate that the most crucial aspect of software engineering is the gathering of requirements.[6] A succinct and accurate requirements speciation allows the entire organization to have a clear understanding of what the specified product will do. This understanding requires four steps. First, the organization must write correct requirements. Second, the organization must ensure that a diverse committee reviews requirements. Third, the requirements must be kept up to date if any changes are required, and requirements *will* change.[7] [20] Fourth and finally, the requirements must then be adhered to from implementation to completion of the project.

At VLO, the addition of the user interface design team, previously discussed in Section 4.2.5, was a factor in the outcome of Pj6. The user interface design team was new to VLO and initially produced requirements that were unsatisfactory, a clear indication that VLO does not effectively perform requirements management. To start, the user interface design team was developing the user interface specification in isolation without the input from other groups that had years of experience working on previous projects, leading to a specification that produced an ineffective user interface. Under these circumstances, it is desirable to ensure that a representative from each of the various groups, including

management, is present when writing requirements, especially given that this team was new to VLO. One way to achieve this representation is Joint Applications Development (JAD), which was created by Drake, Josh, and Crawford at IBM.[1] There is no evidence to suggest that JAD will always work, but studies by Liou et al indicate that JAD and other similar methods do increase the effectiveness of software engineering, especially requirements specification.[14] However, given that the user interface design team was new to VLO, it probably would have been better to involve a cross functional group instead of having the new team dictate the requirements. Finally, the requirements must also be written in accordance with the project schedule, as late requirements are less useful and probably harmful to the project's schedule. An appropriate amount of time must be allocated for requirements engineering, which should always be performed at the beginning of a project lifecycle.[9]

In order to have effective requirements management, requirements need to be researched and documented prior to the implementation stage. If the requirements for a feature in a product are not written before product development begins, then the feature should not be included in the product. Teams should also immediately remove features whose requirements are incomplete from a product, as these features cannot be properly implemented. Building a prototype is an effective method to mock-up features during or prior to the requirements elicitation. However, the caveat is that these prototypes should be done at an early stage of the product and should be immediately discarded once the requirements have been gathered, as the purpose of the prototype has been fulfilled. After the requirements have been elicited, a requirements specification document should be carefully written succinctly describing all the requirements for the project in an unambiguous fashion. This document will define the product and allow project schedules to be created based on the effort estimates of the requirements.

5.4 Maximal Use of Resources

Since VLO is a very large organization and employs a large number of people, all of whom are the most valuable resources, it is vital to maximize the use of these resources. Maximizing the use of people does not imply simply maximizing the number of people. VLO must effectively and efficiently use its people. VLO should not simply hire new employees and place them on a critical path of a product, as discussed with the user interface design team in Section 4.2.5. This is not effective people management. There are solutions to overcome Brooks law, such as utilizing extreme programming,[13] however, in this case it could have been much more beneficial to simply create a cross functional team or even team up employees from different departments.

VLO should attempt to utilize the available resources by conducting pre-release alpha and beta programs within the organization, such programs have been shown to have a profound effect on pre-release software and production software, if the programs are correctly conducted.[28] Conducting controlled experiments with employees, or even potential customers, will produce results. These programs ensure that a small representative population has actually used a product and therefore can provide feedback that should help improve the product before it is released. Note that it is imperative for alpha and beta programs to be correctly administered in order to obtain useful results. Adding incentives for active participation or removing privileges for inactive participation is an effective way to encourage people to actively participate and provide essential feedback.

Finally, there are indications that VLO does not fully understand expanding market demands. Since the marketing teams effectively decide the direction of future products, it is important to perform adequate market research and employ methodologies to ensure the right product is chosen to develop.[31] However, no matter the amount of market research performed, it is vital to have a process in order to achieve the vision set out by the marketing teams. Problems may arise with the marketing teams that are technical in nature and prohibit the product from performing in the desired manner; therefore these cannot be committed to a project. The disconnect that may exist between development and marketing teams needs to be minimized by partnering marketing experts with technical experts in order to achieve a mutual decision for the direction of a product. The importance of leveraging this knowledge transfer has been well documented.[30] With teamwork, the marketing demands can be technically reviewed before being committed and will ultimately be better positioned to succeed.

5.5 Development Order and Priority of Vital Features

None of Pj1, Pj2, Pj3, Pj4, Pj5, and Pj7 contained a revolutionarily new hardware component; each contained new features and enhancements, none of which could be considered trivial, but were not revolutionarily new. Pj6 was considered revolutionary with respect to both hardware and software, but it also included various new features and additional new hardware components. That being said, the focus of Pj6 was not clear. Even though VLO has previously shown that it can incrementally produce products with feature enhancements and new hardware components, VLO did not continually produce revolutionarily new products. Generally, it is not wise to overcommit, especially under extremely tight immovable deadlines. It is far better to concentrate on one key aspect. This concentration is vital when a new revolutionarily component is being developed. Under these circumstances, it would have been better for VLO to focus the majority of its available resources on the revolutionarily new hardware component.

A concentrated focus would give the revolutionarily new hardware component the best chance of achieving its potential.

The feature enhancements and other new hardware components committed to Pj6 were not major selling points and thus should have been deferred. Another option would have been to split the project and stagger the releases into sprints, so that the most important features were developed and released first, these sprints would be consistent with the findings of Molokken-Ostvold et al.[11] Each sprint would have allowed VLO to concentrate on one aspect of the project and ensure that it was done right the first time. That the project continued its development for an additional year after the initial release indicates that although features were completed, the quality was not satisfactory. Given the number of new features in the product, if VLO had concentrated on a select few features then each feature would be higher quality than what was initially released, even though fewer features would be included in the initial release. Only once the product had launched should additional features be considered on an individual basis in maintenance releases. If the initial launch did not meet expectations, then VLO could determine whether there is merit in pursuing additional features. If VLO had enough resources, other features could have been developed in parallel and committed to the project, only when completed. Ultimately, the key is to minimize the number of features and then concentrate on only the core features deemed absolutely necessary.

5.6 Schedule Alignment

Effective requirements management and prioritization of features, as previously discussed in Sections 5.3 and 5.5, should enable VLO to be able to align a project schedule within customer expectations. Without effective requirements management and feature prioritization, VLO was unable to make accurate estimates and undoubtedly committed to implement more features than can realistically be accomplished within the committed schedule. In essence, VLO should ensure that accurate and detailed estimates can be provided for each requested feature, if it is not possible then the feature should not be committed to the project. There are a variety of cost estimation tools, such as the Constructive Cost Model (COCOMO II)[26] originally developed by Barry Boehm, or Wideband Delphi, PROBE, The Planning Game described in “Applied Software Project Management”.[25] However, estimating the cost of a revolutionarily new component is more difficult. Thus, if a revolutionarily new hardware component is being added, then a sufficient amount of prototyping must have occurred beforehand in order to form an estimate. As well, launch date of the project and the features that must be included should be explicitly defined. It is not uncommon for a customer to demand many more features than can be accomplished before the requested

launch date; in this case VLO needs to renegotiate with customers, adjust the customers' expectations, or simply walk away if the demands are unreasonable. The alignment of schedules with realistic customer demands is a vital negotiation tool that must be implemented in order to achieve a desirable and more importantly successful outcome.

It is also important to align the schedule with the stages of the typical software engineering lifecycle. These stages include requirements engineering, design, implementation, and testing. Szoke found that there is a great amount of emphasis on project scheduling, but the methods "used are imprecise and require significant manual efforts."[27] Szoke developed a method that "can significantly accelerate project scheduling production (>50%) and project realization (>10%) through automated schedule production."[27] Alignment is especially important with the requirements engineering stage, which specifies the software requirements. It is impossible to design, implement, or test software if the requirements are incomplete. A common misconception is that delaying the start of implementation by two months, for example, implies that the completion of implementation will be delayed by more than two months. Delaying the implementation in order to fully complete the requirements will ultimately result in a much smoother implementation. Thus, the schedule needs to align with the completion of the requirements engineering phase and other phases should not be started until the previous stage is complete. Finally, as DeMarco stated "all projects that finish late have this one thing in common: they started late",[4] the project schedule must be realistic. If a project will take one year to develop, then the release date must be at least one year away.

Chapter 6
FUTURE & RELATED WORK

6.1 Related Work

There has been a lot of research on a variety of issues discussed in this thesis, including but not limited to requirements engineering, prototyping revolutionary products, user experiments, and prioritizing and schedule projects. However, there appears to be a gap that needs to be bridged between research organizations and commercial organizations.[32] One of the reasons for the gap is the lack of external validity that controlled experiments exhibit.[19] This thesis contains a large amount so this thesis can serve as a starting point for a wide variety of potential further research. However, first it is vital to look at what past related research has been performed on areas from this thesis.

Ellis and Berry are a prime example of bridging the gap between research organizations and commercial organizations. Ellis and Berry demonstrate why empirical evidence is hard to come by and the important role that empirical evidence plays in validating results from research organizations. In particular, Ellis and Berry concentrate on the requirements engineering field. The results are based on the studies performed by Ellis and Berry and previous empirical research work by Hoffman et al,[33] Paech et al,[34] Curtis et al,[35] and Lubars et al.[36] The main contributions from Ellis and Berry are to the requirements engineering process, especially with respect to requirements discovery and management (RDM) and introduce an RDM maturity model along with "six RDM capability areas and a technique for assessing an organization's scores in these areas".[19]

Coldrick et al wrote a paper "A Decision Framework for R&D Project Selection"[31] which is entirely applicable to VLO. Coldrick et al state that the development of revolutionary products is essential to organizations in order to maintain a competitive advantage, which emphasizes the importance of good research and development. Next Coldrick et al indicate that R&D budgets are usually vastly large and the number of possible products to research is practically limitless. Therefore, it is a critical for the organization to select optimal projects to pursue. Coldrick et al "present a project selection model that recognizes the different types of R&D projects and their particular requirements."[31] The model is applicable to projects and organizations of all sizes, so there is a definite relevance to this thesis. The ultimate goal of the model is "provide a formal, structured methodology to assist engineers and management in the decision making process."[31] This model is consistent with the need for VLO to assess

where to focus research and development and when a particular technology has been prototyped enough and is ready to be committed to an actual product.

Finally, in "The Road Ahead for Mining Software Repositories " by Hassan,[12] Hassan describes how software repositories are being transformed from static to dynamic assets. With this transformation, decision-making processes can be guided based on the dynamic software repositories. This field of study focuses on two primary aspects. First, "the creation of techniques to automate and improve the extraction of information from repositories"[12] and second "the discovery and validation of novel techniques and approaches to mine important information from these repositories."[12]

There are several ways an organization can mine their own software repositories in order to increase effectiveness. In particular, there are four related to VLO. The first use is to help organizations better understand large software systems. This can be achieved by looking through historical information about a project, for example in order to determine a particular design decision was made. Crubranic et al describe a tool, Hipikat, which "provides developers with efficient and effective access to the group memory for a software development project that is implicitly formed by all of the artifacts produced during the development."[38] The second use better enables an organization to effectively manage change propagation, which plays a vital role in large software systems that contain many coupled components. By using data mined from the software repository, an organization can predict what components may be affected by a particular change. Hassan and Holt have developed a several heuristics to predict such change propagation and show that "historical dependencies outperform traditional dependency information when propagating changes."[39] The third use is in predicting and identifying bugs. This is accomplished by cross referencing source control repositories with defect tracking repositories, as a Graves et al has found the biggest predictor of defects is previous defects and previous code changes.[37] However, there are static code analysis tools, such as Coverity,[41] that can be used to predict where defects may occur.[40] The fourth use is to mine unofficial methods of communication, such as instant messaging or e-mails. By gathering this information, an organization can determine when to add new developers to a project and to discover the overall morale of employees. Hassan and Rigby used the content of mailing lists to determine the level of developer morale before and after a release, the results of "shows promise in understanding why developers join and leave a project."[42] There is a wealth of untapped information in software repositories at VLO that when mined could contribute to the success of future products.

6.2 Future Work

There are three areas that merit for more research. The first is to analyze the data that were not considered in this thesis. The second is to find criteria for determining when enough research has been done about a new hardware component. Finally, the CR process can be analyzed to improve or otherwise enhance the CR process.

6.2.1 Further Analysis of Data

There are data collected but not analyzed that would benefit from research in the mining repositories field.[12] The data that are most interesting are SCs and customers' acceptances. As previously discussed, a SC may represent a feature request, a CR, a software bug, or a non-functional change such as optimization or change to documentation. This thesis considers all SCs to be of equal value, so there is the potential to study the distribution of SCs and SC types across the various projects. A new study would determine whether a particular type of SC was more prominent in any one particular project. Some questions of interest are: Were particular SCs directly associated with one particular piece of hardware in a particular project? How many SCs were regressions and related to previous SCs?

Measuring and monitoring SCs versus time is also interesting. It can even be further expanded to monitor changes of any aspect of the software engineering process, in particular requirements. The question is whether changes can be tracked and be used to effectively determine a criterion that indicates when a particular module is mature enough and the next stage of the project can commence. For example, determining when to switch from the requirements phase to the development phase is a difficult and important decision. Ideally, the switch should be made when the requirements specification can be implemented without any further clarification. However, if the changes to the requirements specification are accurately tracked, then audits can be performed, and the criterion evaluated in order to determine whether a requirements specification is mature enough to proceed with implementation. This criterion does not imply that no changes to the requirements specification will occur; it simply indicates when the next phase can be initiated. A modified criterion can be applied to SCs and test plans respectively for the development phase and for the verification phase.

The next kind of data for further research is the customer acceptance data. The customer acceptance database is accurate and detailed and contains an exhaustive history. This thesis examines the relationship between the customer acceptance data and the number of completed SCs by that date. However, a study relating customer acceptances with specific hardware and customer timelines is of

notable interest. Such a study would help determine whether a particular piece of hardware was given priority over another by VLO. This study could potentially uncover other problematic areas, especially if a customer acceptance was received for the same product and the same piece of hardware multiple times. Multiple customer acceptances for the same product and hardware platform would be of interest to determine whether there is an underlying reason for the multiple customer acceptances. It is also possible there was a particular issue that the root cause was difficult to determine. A study that explores these issues may uncover some solutions that could help VLO in the future. Finally, on occasion a customer acceptance comes with a condition from the customer. An analysis of the conditions is another area for future research.

6.2.2 Research and Planning

The study of this thesis uncovered a lot of data that question whether sufficient research and planning for revolutionarily new features was performed. When enough research has been done is difficult to determine. Many an organization performs only a small amount of research before committing a revolutionarily new feature to a product. Thus this topic is in dire need of future research. In some circumstances, an organization, particularly a start-up, may have no other option. According to Lewis et al, it is clear that commercial organizations are racing to keep up with the revolutionary changes in just about everything, which is clearly not representative of where the academics have been focusing.[8] However, a mature organization, such as VLO, should be able to devote enough resources for the necessary research.

To start, a preliminary survey should be developed in order to determine how much research other companies perform when developing revolutionarily new features. Next, based on the preliminary survey, particular organizations should be chosen to assess the amount of prior research they performed. Such a study would benefit multiple competing organizations, as it would effectively enable an organization to optimize the amount of research on any one particular revolutionarily feature while all but guaranteeing a successful outcome. Finally, the results would be a set of criteria that would enable an organization to make an informed decision of when a feature is ready to be committed would be extremely beneficial to a multitude of organizations.

Another potential area of future work is to develop contingency plans for an organization that does not perform enough research. Some of the questions that should be researched are: What kind of planning can an organization do ahead of time in order to make up time lost in the future due to lack of research? Is there a way for an organization to successfully deploy a premature project? These questions are not

easy to answer, but they may provide some possible solutions to problems faced by many organizations today.

Finally, there are a multitude of studies that indicate that project estimates do not accurately represent the actual amount of work required, especially with respect to software.[16] [17] [18] It is conceivably even harder to provide estimates for technologies that are yet undeveloped and immature. Hence, it is worth studying how to plan and provide estimates for revolutionarily new technologies.

6.2.3 Change Requests

CRs are an important aspect of requirements engineering. Although this thesis does not reveal any problems per se with the CR practices at VLO, there is still a lot of information regarding the CRs and the CR process could be analyzed. For example, a month can elapse from the time a CR is submitted until the CR is approved for implementation. This month is a relatively large duration for a project whose duration is less than a year. Therefore, is it possible to make the CR process more efficient?

Another possible study is to find a method to determine whether a CR is worth the extra effort. The ultimate goal of this study would be to develop criteria that enable an organization to decide whether to approve a CR based on the history of previous CRs. If successful, the results of such a study could reduce the number of CRs.

Finally, another possible study is to develop a method to determine whether all impacted teams have been mentioned in a CR. When a CR is submitted it includes estimates for development, testing, and lists other parts of the organization thought to be affected by the CR. It is very difficult but important to ensure that a CR has been reviewed by all of its impacted teams. If a CR with an incomplete impact analysis is accepted, then the full impact of the CR cannot be assessed. Occasionally, a CR has been accepted and development work has started, but some time later, unexpected impacts are discovered that cause delay. One possible solution to this problem is having an up-to-date trace matrix to determine the actual impact of any change. Ultimately, any improvement to the CR process is beneficial.

Chapter 7
CONCLUSIONS

VLO allowed this author access to resources for analysis. Seven different products, Pd1, Pd2, Pd3, Pd4, Pd5, Pd6, and Pd7, were selected to study for this thesis. Each product had its own development project, Pj1, Pj2, Pj3, Pj4, Pj5, Pj6, and Pj7, which provided the data for analysis. First, this thesis describes and categorizes each of the seven products. Next, a set of research questions are posed that guide the analysis throughout this thesis. The questions are listed along with the section where the answer can be found:

RQ1: What caused Pd6 to fail? (Section 4.2)

RQ2: What indications were there during Pd6's development that Pd6 would fail? (Section 4.2)

RQ3: What could VLO have done in order to prevent Pd6 from becoming a failure? (Chapter 5)

RQ4: What can VLO learn from the failure? (Chapter 5)

RQ5: Are there indications from Pd1, Pd2, Pd3, Pd4, Pd5, or Pd7 that indicate areas that VLO is excelling at? (Section 4.1.7)

The data extracted from the projects include the number of source commits, the number of change requests, number of customers' acceptances, various project dates, employee opinions, and a variety of product and project documents from a database.

The data for Pj1, Pj2, Pj3, Pj4, Pj5, and Pj7 were first analyzed and any indications of the reasons these projects had successful outcomes were noted. Next, the data for Pj6 were analyzed and compared with those of the six other projects. Several intriguing factors were discovered about Pj6. To start, there was the addition of new hardware to Pd6. Pd6 included a revolutionarily new hardware component plus other additional components that had not been included in any previous product. These additional hardware components both caused severe delays in the delivery of the hardware to other teams. These delays also had an effect on the software, much of which depends directly on the hardware platform. Finally, during the development of Pd6, a new user interface design team was given the sole responsibility of designing the user interface specification for this project from the ground upwards. All of these factors contributed to the poor outcome of Pj6.

There are six possible lessons to learn from Pj6 for VLO. First, VLO must not release hardware until its major issues are resolved. Unfortunately, sometimes these issues may not arise until the hardware and the software have been integrated. Second, VLO must decouple the hardware and the software as much as possible, which would help to avoid as much dependency as possible. Third, VLO must ensure that an

effective and efficient process is in place in order to manage the requirements across each project. Fourth, VLO must maximize the use of its resources, especially employees, to ensure the right product is being built. Fifth, VLO must accurately prioritize features to ensure that the quality of the essential features is not sacrificed in order to add non-critical features. Sixth and final is that VLO must align the schedules across the organization to ensure that each and every employee is aware and working towards the schedule. Applying these six lessons to a project does not guarantee the outcome of the project will be a success. However, it will undoubtedly help VLO to succeed in the long run.

There are three areas of interest for future research. First, there are an abundance of data that have not been studied. These data include source commits and customers' acceptances. Second, development of criteria for deciding whether enough research in a revolutionarily new hardware platform has taken place in order to advance to the next development stage. Finally, the change request process could benefit from some in-depth research.

The overall underlying message throughout this thesis is that inadequate requirements engineering in a project can and will plague the entire duration of the project. It is not necessarily true that delaying the commencement of the implementation phase of development will cause an equal or greater delay in the completion of the implementation phase. Many organizations are under pressure to deliver products by a deadline, so they tend to start the implementation before the requirements engineering is complete. The requirements engineering phase needs to be completed prior to the start of implementation, otherwise the implementers will not know exactly what to implement. It is equivalent to starting the construction of a building without a complete set of blue prints.

However, even with proper requirements engineering, the requirements specification could still very well be wrong. This situation occurred twice with Pj6, once with the revolutionarily new hardware component and once again with the user interface design. However, even with requirements engineering, each of these situations did not have desirable outcomes due to the fact that the requirements were wrong. Therefore, it is vital that VLO devote enough time to perform adequate prototyping, researching requirements, and ultimately writing requirements specifications.

Appendix A:
Collected Data

This appendix contains the compiled data collected and referenced in this thesis, including data from the source control repositories, change request reports, and customers' acceptances for all the products discussed.

Project 1

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	2	0	0.04	0.04	0	0.00	0.00
2	11	0	0.21	0.25	0	0.00	0.00
3	20	0	0.39	0.64	0	0.00	0.00
4	16	0	0.31	0.95	0	0.00	0.00
5	82	0	1.58	2.53	0	0.00	0.00
6	1218	44	23.50	26.02	0	0.00	0.00
7	1661	12	32.04	58.06	0	0.00	0.00
8	746	9	14.39	72.45	6	0.85	0.85
9	489	7	9.43	81.89	26	3.70	4.56
10	406	2	7.83	89.72	58	8.26	12.82
11	198	5	3.82	93.54	72	10.26	23.08
12	150	1	2.89	96.43	52	7.41	30.48
13	71	0	1.37	97.80	47	6.70	37.18
14	47	1	0.91	98.71	114	16.24	53.42
15	13	2	0.25	98.96	71	10.11	63.53
16	20	2	0.39	99.34	41	5.84	69.37
17	11	0	0.21	99.56	32	4.56	73.93
18	1	0	0.02	99.58	43	6.13	80.06
19	11	0	0.21	99.79	28	3.99	84.05
20	1	0	0.02	99.81	27	3.85	87.89
21	7	0	0.14	99.94	24	3.42	91.31
22	1	0	0.02	99.96	16	2.28	93.59
23	0	0	0.00	99.96	13	1.85	95.44
24	0	0	0.00	99.96	14	1.99	97.44
25	0	0	0.00	99.96	2	0.28	97.72
26	2	0	0.04	100.00	7	1.00	98.72
TOTAL:	5184	85	100.00		693	98.72	
				Total Acceptances[1]:	702		

Table 3: Project 1 Data

[1] The number of acceptances is only shown until the end of the development schedule, which explains why the total number of acceptances is higher than the total number in the table.

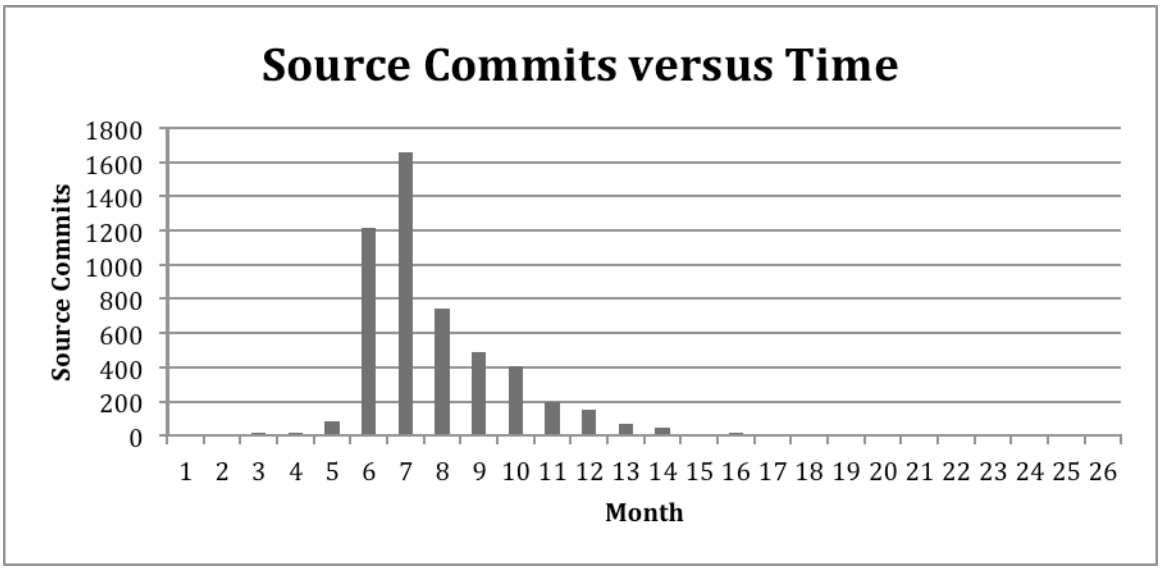

Figure 3: Project 1 Source Commits versus Time

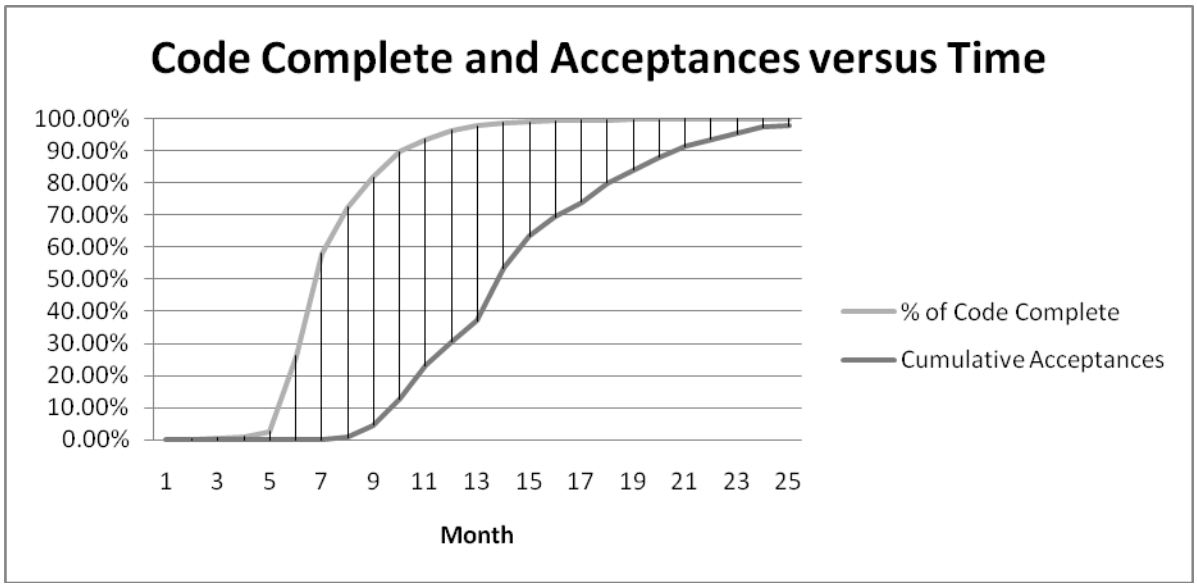

Figure 4: Project 1 Code Complete and Acceptances versus Time

Project 2

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	1	0	0.01	0.01	0	0.00	0.00
2	1368	8	16.08	16.09	0	0.00	0.00
3	1925	6	22.63	38.72	0	0.00	0.00
4	1412	10	16.60	55.31	0	0.00	0.00
5	1224	5	14.39	69.70	0	0.00	0.00
6	824	4	9.69	79.38	0	0.00	0.00
7	746	7	8.77	88.15	5	1.98	1.98
8	348	6	4.09	92.24	13	5.14	7.11
9	269	7	3.16	95.40	32	12.65	19.76
10	125	4	1.47	96.87	24	9.49	29.25
11	143	5	1.68	98.55	40	15.81	45.06
12	84	0	0.99	99.54	38	15.02	60.08
13	17	0	0.20	99.74	30	11.86	71.94
14	6	0	0.07	99.81	33	13.04	84.98
15	12	0	0.14	99.95	14	5.53	90.51
16	0	0	0.00	99.95	8	3.16	93.68
17	1	0	0.01	99.96	7	2.77	96.44
18	3	0	0.04	100.00	2	0.79	97.23
TOTAL:	8508	62	100.00		246	97.23	
				Total Acceptances[1]:	253		

Table 4: Project 2 Data

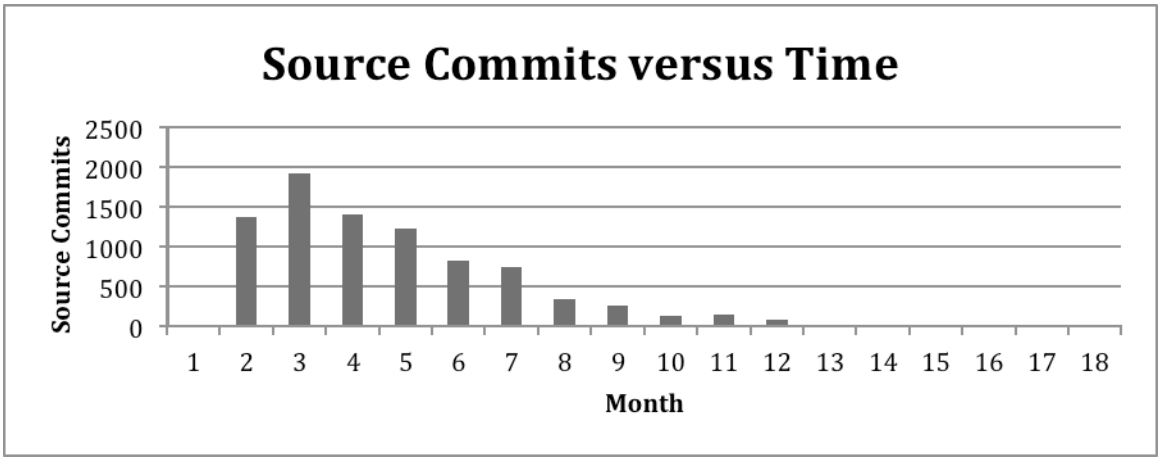

Figure 5: Project 2 Source Commits versus Time

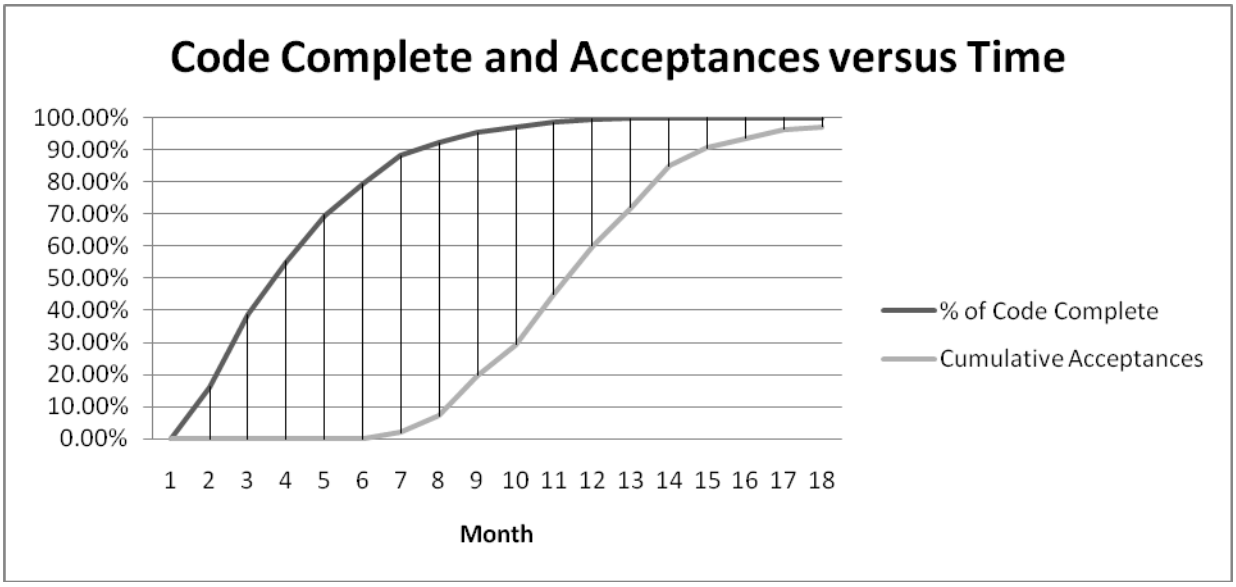

Figure 6: Project 2 Code Complete and Acceptances versus Time

Project 3

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	462	9	6.43	6.43	0	0.00	0.00
2	440	7	6.12	12.55	0	0.00	0.00
3	979	8	13.63	26.18	0	0.00	0.00
4	1267	13	17.63	43.81	0	0.00	0.00
5	767	8	10.68	54.49	0	0.00	0.00
6	364	8	5.07	59.55	0	0.00	0.00
7	985	3	13.71	73.26	0	0.00	0.00
8	515	4	7.17	80.43	0	0.00	0.00
9	254	2	3.54	83.97	0	0.00	0.00
10	284	7	3.95	87.92	16	0.77	0.77
11	184	3	2.56	90.48	86	4.11	4.88
12	145	2	2.02	92.50	175	8.37	13.25
13	80	1	1.11	93.61	191	9.14	22.39
14	104	2	1.45	95.06	184	8.80	31.20
15	73	1	1.02	96.08	122	5.84	37.03
16	48	4	0.67	96.74	64	3.06	40.10
17	49	5	0.68	97.43	107	5.12	45.22
18	36	3	0.50	97.93	86	4.11	49.33
19	28	3	0.39	98.32	66	3.16	52.49
20	30	0	0.42	98.73	84	4.02	56.51
21	10	0	0.14	98.87	49	2.34	58.85
22	7	1	0.10	98.97	29	1.39	60.24
23	11	0	0.15	99.12	90	4.31	64.55
24	10	0	0.14	99.26	52	2.49	67.03
25	4	0	0.06	99.32	49	2.34	69.38
26	16	0	0.22	99.54	90	4.31	73.68
27	4	0	0.06	99.60	178	8.52	82.20
28	6	0	0.08	99.68	108	5.17	87.37
29	2	0	0.03	99.71	84	4.02	91.39
30	21	0	0.29	100.00	51	2.44	93.83
TOTAL:	7185	94	100.00		1961	93.83	
				Total Acceptances[1]:	2090		

Table 5: Project 3 Data

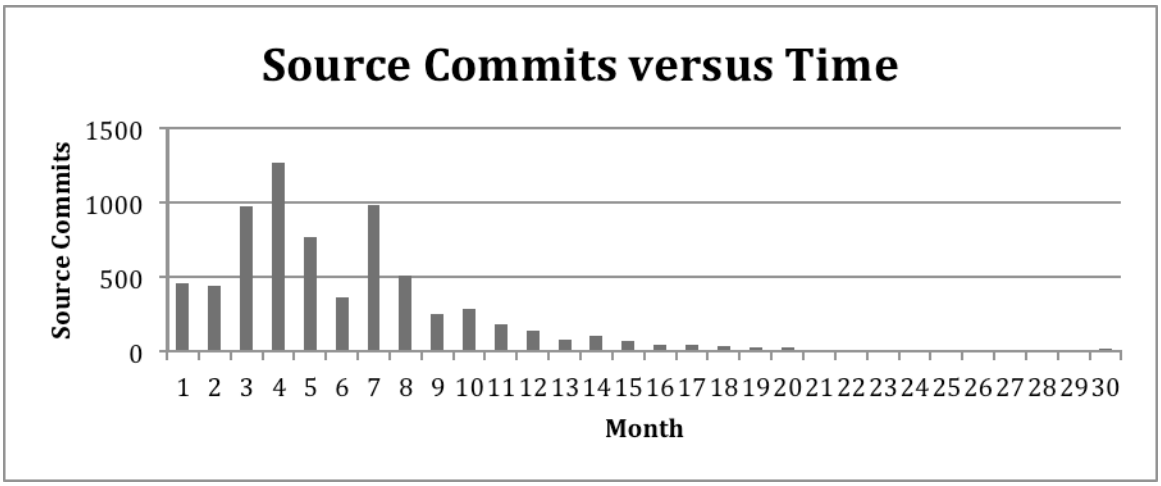

Figure 7: Project 3 Source Commits versus Time

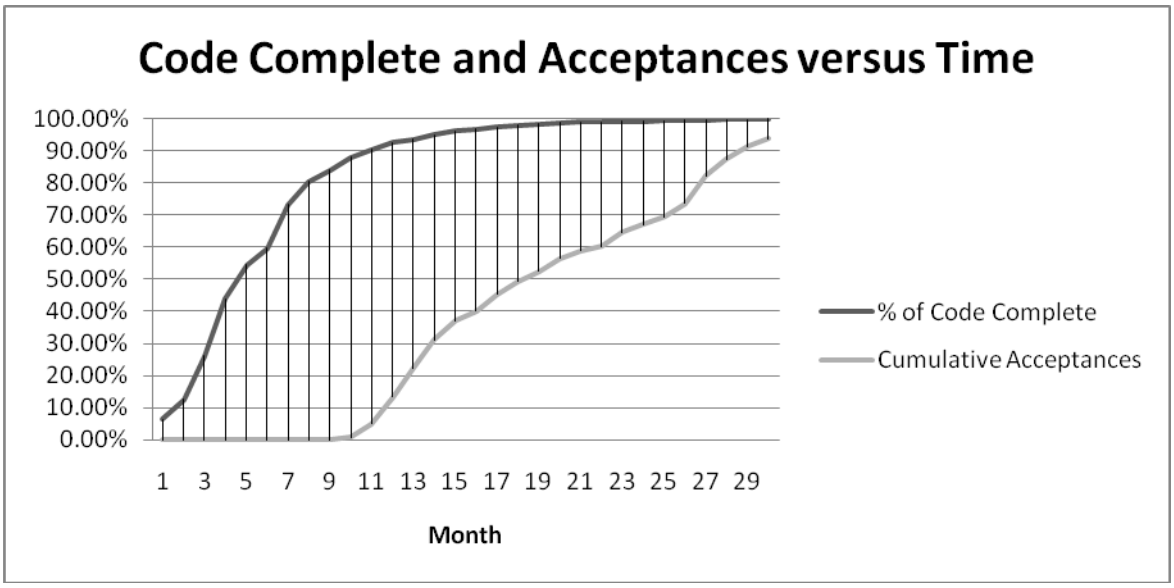

Figure 8: Project 3 Code Complete and Acceptances versus Time

Project 4

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	555	0	4.35	4.35	0	0.00	0.00
2	1885	21	14.76	19.11	0	0.00	0.00
3	1543	24	12.09	31.20	0	0.00	0.00
4	1436	21	11.25	42.45	0	0.00	0.00
5	1553	10	12.16	54.61	0	0.00	0.00
6	1259	24	9.86	64.47	0	0.00	0.00
7	967	26	7.57	72.05	24	2.08	2.08
8	760	15	5.95	78.00	92	7.99	10.07
9	564	16	4.42	82.42	63	5.47	15.54
10	604	9	4.73	87.15	134	11.63	27.17
11	578	20	4.53	91.67	60	5.21	32.38
12	246	5	1.93	93.60	57	4.95	37.33
13	237	6	1.86	95.46	64	5.56	42.88
14	133	4	1.04	96.50	60	5.21	48.09
15	126	5	0.99	97.49	76	6.60	54.69
16	112	3	0.88	98.36	76	6.60	61.28
17	69	1	0.54	98.90	40	3.47	64.76
18	21	2	0.16	99.07	46	3.99	68.75
19	43	2	0.34	99.40	42	3.65	72.40
20	53	1	0.42	99.82	29	2.52	74.91
21	3	0	0.02	99.84	34	2.95	77.86
22	4	0	0.03	99.87	114	9.90	87.76
23	0	0	0.00	99.87	57	4.95	92.71
24	0	0	0.00	99.87	31	2.69	95.40
25	16	0	0.13	100.00	20	1.74	97.14
TOTAL:	12767	215	100.00		1119	97.14	
				Total Acceptances[1]:	1152		

Table 6: Project 4 Data

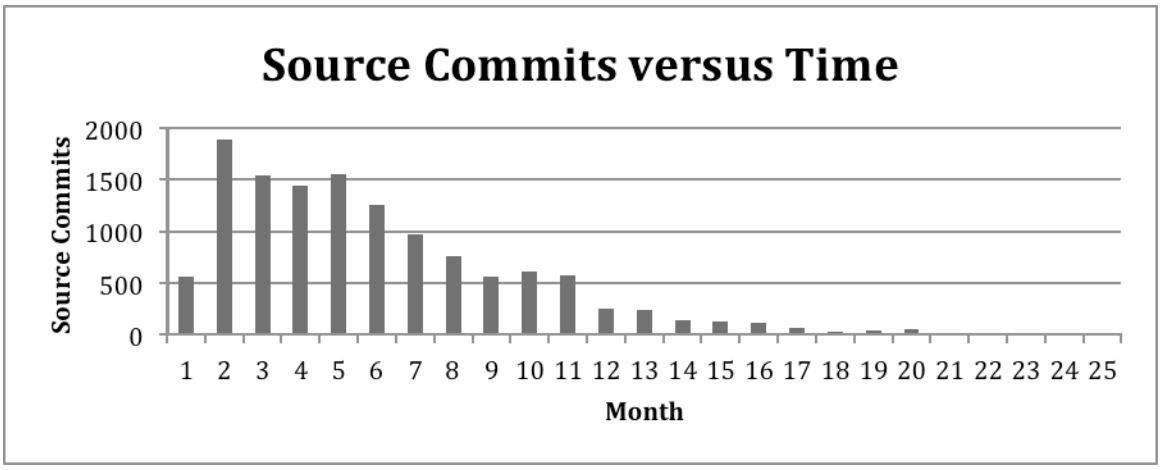

Figure 9: Project 4 Source Commits versus Time

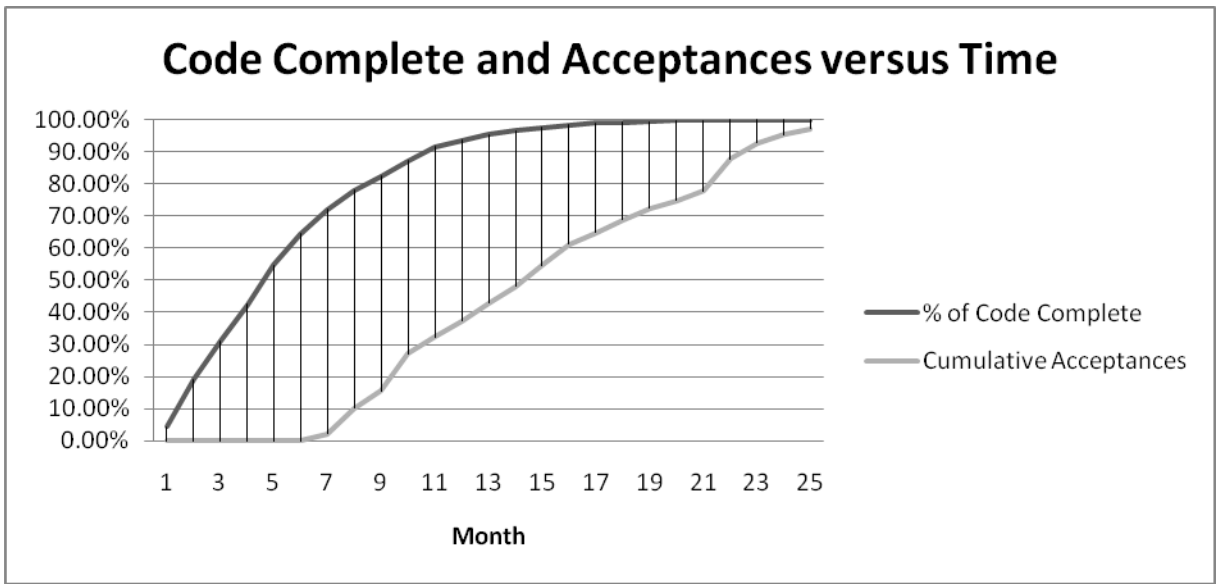

Figure 10: Project 4 Code Complete and Acceptances versus Time

Project 5

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	0	1	0.00	0.00	0	0.00	0.00
2	1	0	0.01	0.01	0	0.00	0.00
3	211	0	2.65	2.66	0	0.00	0.00
4	1391	5	17.48	20.15	0	0.00	0.00
5	1159	8	14.57	34.72	0	0.00	0.00
6	1086	6	13.65	48.37	0	0.00	0.00
7	540	16	6.79	55.15	0	0.00	0.00
8	846	9	10.63	65.79	26	2.20	2.20
9	409	2	5.14	70.93	72	6.08	8.28
10	229	15	2.88	73.81	52	4.39	12.67
11	247	9	3.10	76.91	79	6.67	19.34
12	274	10	3.44	80.35	40	3.38	22.72
13	301	8	3.78	84.14	67	5.66	28.38
14	392	3	4.93	89.06	52	4.39	32.77
15	329	6	4.14	93.20	33	2.79	35.56
16	324	4	4.07	97.27	110	9.29	44.85
17	107	1	1.34	98.62	117	9.88	54.73
18	76	1	0.96	99.57	92	7.77	62.50
19	17	1	0.21	99.79	102	8.61	71.11
20	10	0	0.13	99.91	114	9.63	80.74
21	6	0	0.08	99.99	62	5.24	85.98
22	1	0	0.01	100.00	45	3.80	89.78
TOTAL:	7956	105	100.00		1063	89.78	
				Total Acceptances[1]:	1184		

Table 7: Project 5 Data

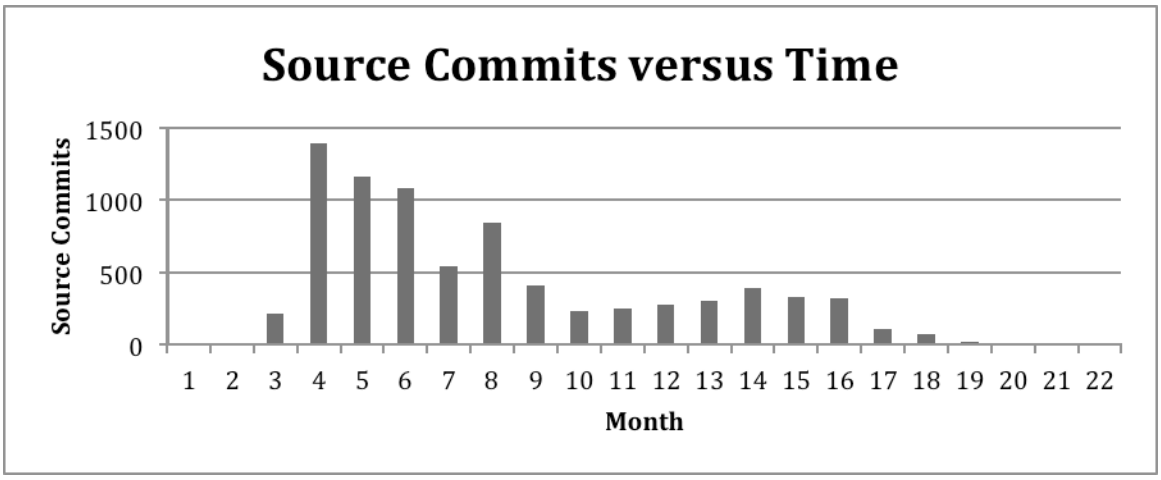

Figure 11: Project 5 Source Commits versus Time

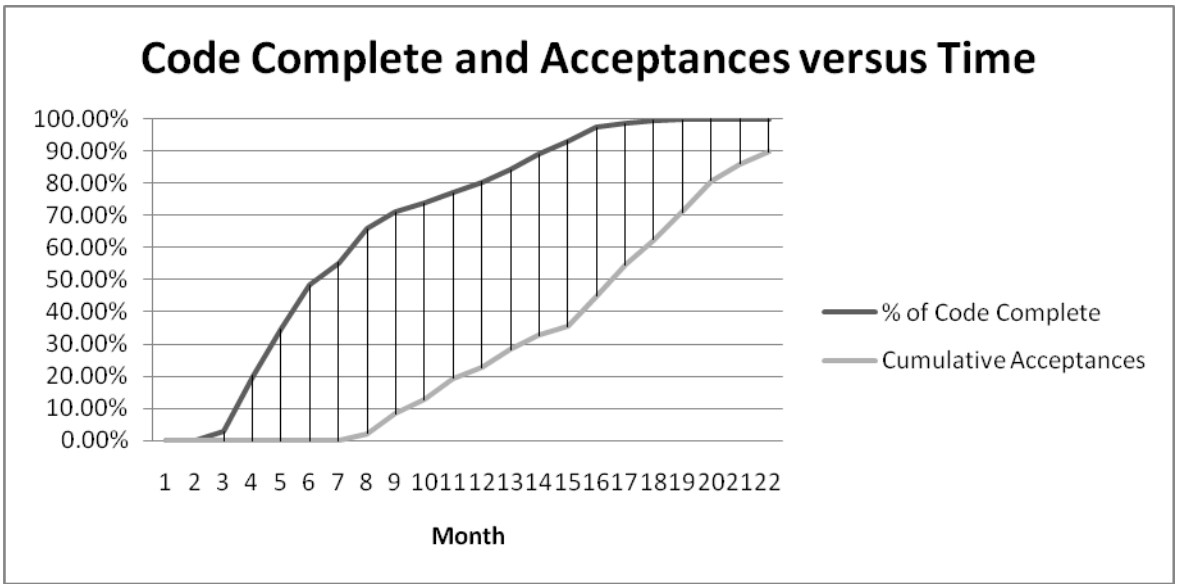

Figure 12: Project 5 Code Complete and Acceptances versus Time

Project 6

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	0	3	0.00	0.00	0	0.00	0.00
2	71	2	0.41	0.41	0	0.00	0.00
3	1664	15	9.64	10.05	0	0.00	0.00
4	2051	8	11.88	21.93	0	0.00	0.00
5	2215	10	12.83	34.76	0	0.00	0.00
6	1344	14	7.79	42.55	0	0.00	0.00
7	1888	9	10.94	53.49	0	0.00	0.00
8	1880	14	10.89	64.38	0	0.00	0.00
9	604	2	3.50	67.88	13	4.21	4.21
10	809	8	4.69	72.56	14	4.53	8.74
11	903	18	5.23	77.80	1	0.32	9.06
12	373	5	2.16	79.96	5	1.62	10.68
13	572	16	3.31	83.27	27	8.74	19.42
14	418	3	2.42	85.69	40	12.94	32.36
15	1074	15	6.22	91.91	20	6.47	38.83
16	775	3	4.49	96.40	13	4.21	43.04
17	417	1	2.42	98.82	13	4.21	47.25
18	135	1	0.78	99.60	55	17.80	65.05
19	58	1	0.34	99.94	30	9.71	74.76
20	10	0	0.06	99.99	28	9.06	83.82
21	0	0	0.00	99.99	25	8.09	91.91
22	0	0	0.00	99.99	14	4.53	96.44
23	1	0	0.01	100.00	8	2.59	99.03
TOTAL:	17262	148	100.00		306	99.03	
				Total Acceptances[1]:	309		

Table 8: Project 6 Data

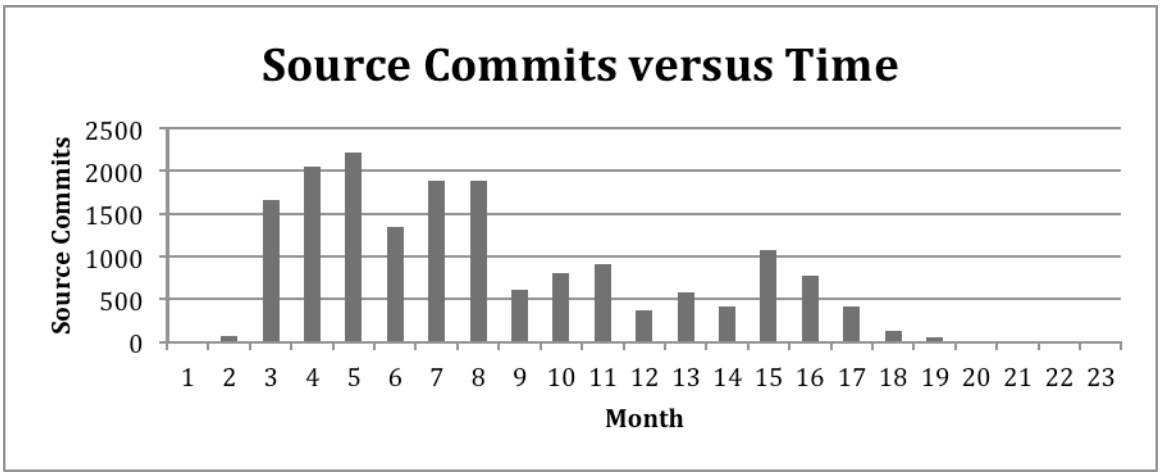

Figure 13: Project 6 Source Commits versus Time

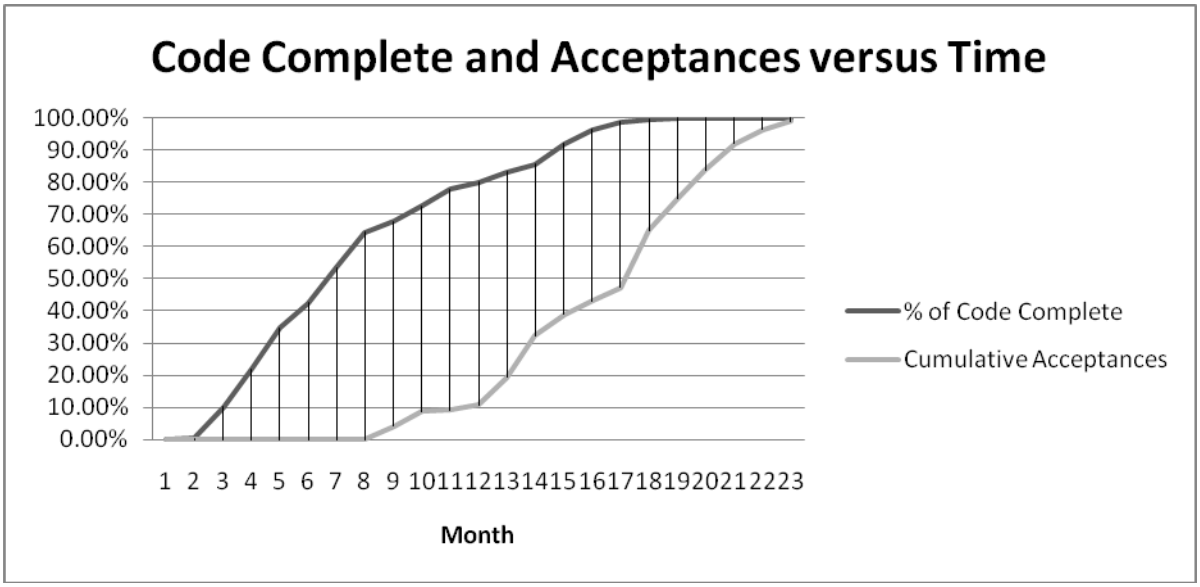

Figure 14: Project 6 Code Complete and Acceptances versus Time

Project 7

Month	Number of Source Commits	Number of Change Requests	Percentage of Source Commits	Percentage Code Complete	Number of Acceptances	Percentage of Acceptances	Cumulative Percentage of Total Acceptances
1	0	4	0.00	0.00	0	0.00	0.00
2	6	3	0.13	0.13	0	0.00	0.00
3	44	8	0.98	1.12	0	0.00	0.00
4	716	16	15.99	17.11	0	0.00	0.00
5	912	8	20.37	37.48	0	0.00	0.00
6	1377	2	30.76	68.24	0	0.00	0.00
7	419	3	9.36	77.60	0	0.00	0.00
8	442	4	9.87	87.47	3	6.52	6.52
9	253	0	5.65	93.12	2	4.35	10.87
10	156	0	3.48	96.60	2	4.35	15.22
11	120	0	2.68	99.29	3	6.52	21.74
12	31	1	0.69	99.98	23	50.00	71.74
13	0	0	0.00	99.98	5	10.87	82.61
14	0	0	0.00	99.98	0	0.00	82.61
15	1	0	0.02	100.00	1	2.17	84.78
TOTAL:	4477	49	100.00		39	84.78	
				Total Acceptances[1]:	46		

Table 9: Project 7 Data

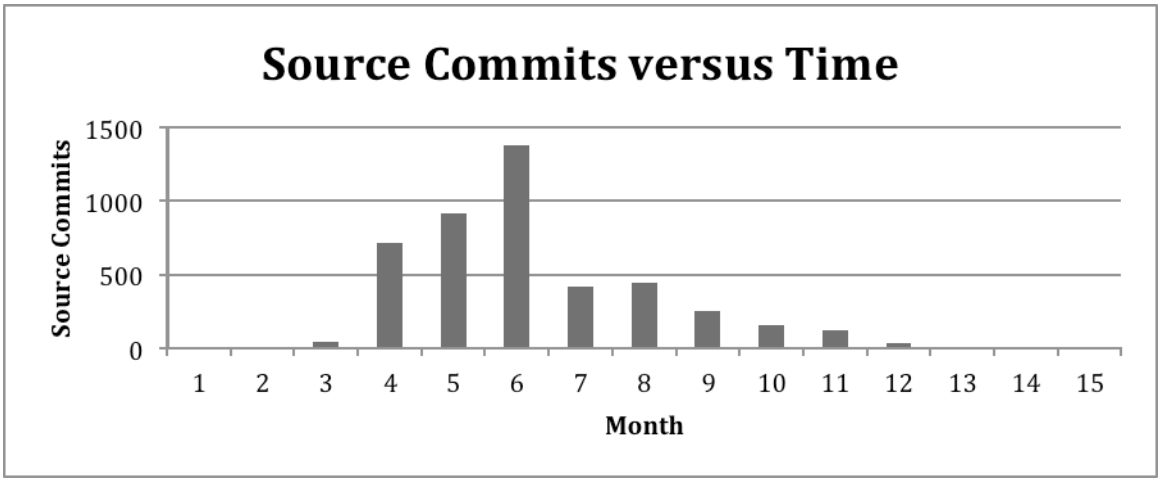

Figure 15: Project 7 Source Commits versus Time

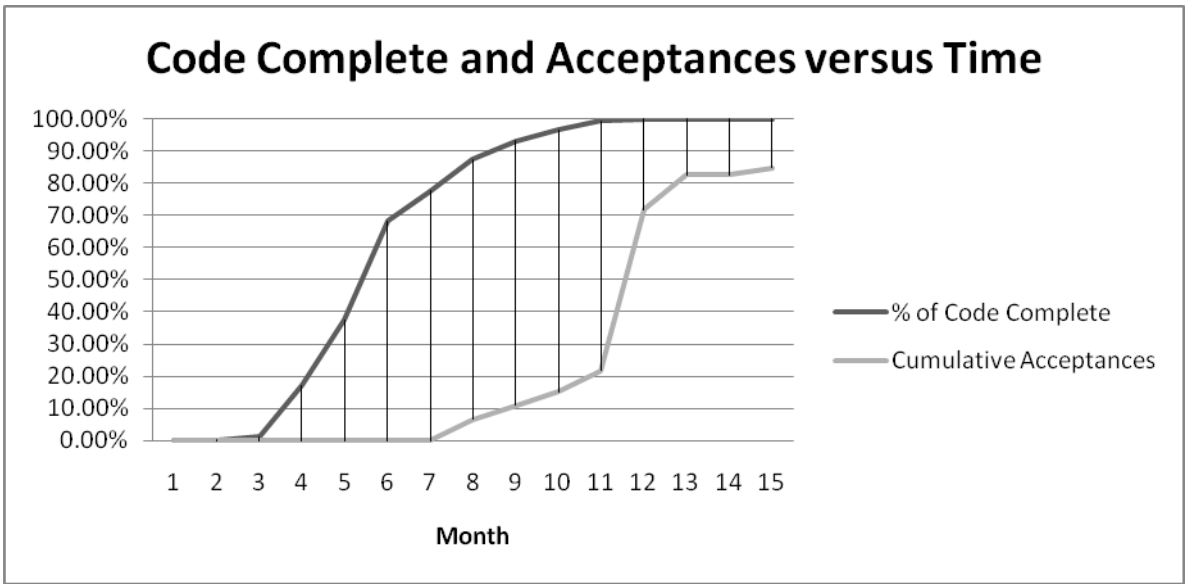

Figure 16: Project 7 Code Complete and Acceptances versus Time

References

1. Joint Applications Development Wikipedia article. URL: http://en.wikipedia.org/wiki/Joint_application_design.

2. Perry, D.E.; Sim, S.E.; Easterbrook, S.M.; "Case studies for software engineers," *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on* , vol., no., pp. 736- 738, 23-28 May 2004; doi: 10.1109/ICSE.2004.1317512; URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1317512&isnumber=29176.

3. Kang, Byung-Kyoo; Bieman, James M.; "A quantitative framework for software restructuring," *Journal of Software Maintenance: Research and Practice*, vol. 11, 1999.

4. DeMarco, Tom; , "All Late Projects Are the Same," *Software, IEEE* , vol.28, no.6, pp.104, Nov.-Dec. 2011; doi: 10.1109/MS.2011.134; URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=6055668&isnumber=6055650.

5. Charette, R.N.; , "Why software fails [software failure]," *Spectrum, IEEE* , vol.42, no.9, pp. 42-49, Sept. 2005; doi: 10.1109/MSPEC.2005.1502528; URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1502528&isnumber=32236.

6. Blyth, A.J.C.; Chudge, J.; Dobson, J.E.; Strens, M.R.; , "A framework for modelling evolving requirements," *Computer Software and Applications Conference, 1993. COMPSAC 93. Proceedings., Seventeenth Annual International* , vol., no., pp.83-89, 1-5 Nov 1993; doi: 10.1109/CMPSAC.1993.404219; URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=404219&isnumber=9093.

7. Kelly, Allan; "Why do requirements change?," *Overload Journal*, vol. 59, Feb 2005; URL: http://accu.org/index.php/journals/319.

8. Lewis, T.; Power, D.; Meyer, B.; Grimes, J.; Potel, M.; Vetter, R.; Laplante, P.; Pree, W.; Pomberger, G.; Hill, M.D.; Larus, J.R.; Wood, D.A.; Weide, B.W.; , "Where is software headed? A virtual roundtable," *Computer* , vol.28, no.8, pp.20-32, Aug 1995; doi: 10.1109/2.402054;

URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=402054&isnumber=9058.

9. Bowles, J.B.; , "Better software reliability by getting the requirements right," *Reliability and Maintainability Symposium, 2006. RAMS '06. Annual* , vol., no., pp.110-115, 23-26 Jan. 2006; doi: 10.1109/RAMS.2006.1677359;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1677359&isnumber=34933.

10. Ruhe, G.; Greer, D.; , "Quantitative studies in software release planning under risk and resource constraints," *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on* , vol., no., pp. 262- 270, 30 Sept.-1 Oct. 2003; doi: 10.1109/ISESE.2003.1237987;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1237987&isnumber=27773.

11. Molokken-Ostvold, K.; Jorgensen, M.; , "A comparison of software project overruns - flexible versus sequential development models," *Software Engineering, IEEE Transactions on* , vol.31, no.9, pp. 754- 766, Sept. 2005; doi: 10.1109/TSE.2005.96;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1514444&isnumber=32435.

12. Hassan, A.E.; , "The road ahead for Mining Software Repositories," *Frontiers of Software Maintenance, 2008. FoSM 2008.* , vol., no., pp.48-57, Sept. 28 2008-Oct. 4 2008; doi: 10.1109/FOSM.2008.4659248;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=4659248&isnumber=4659234.

13. Opelt, K.; , "Overcoming Brooks' Law," *Agile, 2008. AGILE '08. Conference* , vol., no., pp.208-211, 4-8 Aug. 2008; doi: 10.1109/Agile.2008.55;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=4599478&isnumber=4599440.

14. Liou, Y.I.; Chen, M.; , "Integrating group support systems, joint application development, and computer-aided software engineering for requirements specification ," *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on* , vol.iii, no., pp.4-12 vol.3,

5-8 Jan 1993; doi: 10.1109/HICSS.1993.284291;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=284291&isnumber=7027.

15. Grimstad, S.; , "Understanding of estimation accuracy in software development projects," *Software Metrics, 2005. 11th IEEE International Symposium* , vol., no., pp.2 pp.-42, 1-1 Sept. 2005; doi: 10.1109/METRICS.2005.50;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1509320&isnumber=32322.

16. Ferens, D.V.; , "The conundrum of software estimation models," *Aerospace and Electronic Systems Magazine, IEEE* , vol.14, no.3, pp.23-29, Mar 1999; doi: 10.1109/62.750425;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=750425&isnumber=16219.

17. Grimstad, S.; , "Understanding of estimation accuracy in software development projects," *Software Metrics, 2005. 11th IEEE International Symposium* , vol., no., pp.2 pp.-42, 1-1 Sept. 2005; doi: 10.1109/METRICS.2005.50;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1509320&isnumber=32322.

18. Moloekken-OEstvold, K.; Joergensen, M.; Tanilkan, S.S.; Gallis, H.; Lien, A.C.; Hove, S.W.; , "A survey on software estimation in the Norwegian industry," *Software Metrics, 2004. Proceedings. 10th International Symposium on* , vol., no., pp. 208- 219, 14-16 Sept. 2004; doi: 10.1109/METRIC.2004.1357904;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1357904&isnumber=29794.

19. Ellis, Keith; Berry, Daniel M.; "Quantifying the Impact of Requirements Definition and Management Process Maturity on Project Outcome in Business Application Development"; 2011;
URL: http://se.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/EllisBerry.pdf.

20. M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proceedings of the IEEE 68(9), pp. 1060–1076 (September 1980).

21. James W. Hooper and Pei Hsia. 1982. Scenario-based prototyping for requirements identification. In *Proceedings of the workshop on Rapid prototyping*. ACM, New York, NY, USA, 88-93. DOI=10.1145/1006259.1006275 http://doi.acm.org/10.1145/1006259.1006275.

22. Davis, A.M.; , "Operational prototyping: a new development approach," *Software, IEEE* , vol.9, no.5, pp.70-78, Sep 1992; doi: 10.1109/52.156899;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=156899&isnumber=4064.

23. J. Bowers and J. Pycock, "Talking Through Design: Requirements and Resistance in Cooperative Prototyping", pp. 299–305 in Proceedings of the 1994 Computer-Human Interaction Conference (CHI'94), ACM SIGCHI, New York, NY (1994).

24. Power.org, "Debugging Multicore Software Using Virtual Hardware", 2008-05-30, https://www.power.org/events/powercon/paris/Virtutech-Multicore_Debug_PAC_EU_May-2008v6.pdf.

25. Stellman, Andrew, "Applied Software Project Management", Sebastopol, CA : O'Reilly, 2006.

26. COCOMO II, http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html, Accessed December 13, 2011.

27. Szoke, A.; , "A Proposed Method for Automated Project Scheduling using Goals and Scenarios," *International Requirements Engineering, 2008. RE '08. 16th IEEE* , vol., no., pp.339-340, 8-12 Sept. 2008; doi: 10.1109/RE.2008.23
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=4685702&isnumber=4685636.

28. Li, P.L.; Kivett, R.; Zhiyuan Zhan; Sung-eok Jeon; Nagappan, N.; Murphy, B.; Ko, A.J.; , "Characterizing the differences between pre- and post- release versions of software," *Software Engineering (ICSE), 2011 33rd International Conference on* , vol., no., pp.716-725, 21-28 May 2011; doi: 10.1145/1985793.1985894;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=6032513&isnumber=6032438.

29. Marczak, S.; Kwan, I.; Damian, D.; , "Investigating Collaboration Driven by Requirements in Cross-Functional Software Teams," *Requirements: Communication, Understanding and*

Softskills, 2009 Collaboration and Intercultural Issues on , vol., no., pp.15-22, 31-31 Aug. 2009 doi: 10.1109/CIRCUS.2009.2;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=5457327&isnumber=5457324.

30. F. Samer and L. Sproull, "Coordinating expertise in software development teams," Management Science, vol. 46, no. 12, pp. 1554–1568, 2000.

31. Coldrick, S.; Lawson, C.P.; Ivey, P.C.; Lockwood, C.; , "A decision framework for R&D project selection," *Engineering Management Conference, 2002. IEMC '02. 2002 IEEE International* , vol.1, no., pp. 413- 418 vol.1, 2002; doi: 10.1109/IEMC.2002.1038468;
URL: http://ieeexplore.ieee.org.proxy.lib.uwaterloo.ca/stamp/stamp.jsp?tp=&arnumber=1038468&isnumber=22257.

32. H. Kaindl, S. Brinkkemper, J. A. Bubenko, Jr., B. Farbey, S. J. Greenspan, C. L. Heitmeyer, J. C. S. P. Leite, N. R. Mead, J. Mylopoulos, and J. I. A. Siddiqi. Requirements engineering and technology transfer: Obstacles, incentives and improvement agenda. Requirements Engineering, 7(3):113–123, 2002.

33. H.F. Hofmann, F. Lehner; Requirements engineering as a success factor in software projects. IEEESoftware, 18(4):58–66, 2001.

34. B. Paech, T. Koenig, L. Borner, and A. Aurum. An analysis of empirical requirements engineering survey data.In Engineering and Managing Software Requirements, Part 3, pages 427–452, Berlin, Germany, 2005. Springer.

35. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. Communications of the ACM, 31(11):1268–1287, 1988.

36. M. Lubars, C. Potts, and C. Richter. A review of the state of the practice in requirements modeling. In Proceedings of the IEEE International Symposium on Requirements Engineering (RE), pages 2–14, 1993.

37. T.L. Graves, A.F. Karr, J.S. Marron, and H.P. Siy. Predicting Fault Incidence Using Software Change History. IEEE Transactions on Software Engineering, 26(7):653–661, 2000.

38. D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A Project Memory for Software Development. IEEE Trans. Software Eng., 31(6):446–465, 2005.

39. A.E. Hassan and R.C. Holt. Predicting Change Propagation in Software Systems. In Proceedings of the 20th International Conference on Software Maintenance, Chicago, USA, Sept. 2004.

40. D. R. Engler, D. Y. Chen, and A. Chou. Bugs as Inconsistent Behavior: A General Approach to Inferring Errors in Systems Code. In SOSP, pages 57–72, 2001.

41. Coverity, http://www.coverity.com.

42. P. C. Rigby and A. E. Hassan. What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. In MSR [4], page 23.