An empirical evaluation of an agile modular software development approach

A case study with Ericsson

Brandon Lake

Department of Computer and Systems Sciences

Degree project 30 HE credits
Computer and Systems Sciences
Degree project at the master level
Spring term 2012
Supervisor: Rahim Rahmani
Reviewer: Paul Johannesson
An empirical evaluation of an agile modular software development approach

A case study with Ericsson

Brandon Lake

Abstract

Software development is a growing part of many businesses today. The software development approach that is used to develop new software may have a positive or negative impact on the efficiency and quality of the final software product. The objective of this thesis was to propose a refined software development approach and to test it empirically. The software development approach is comprised of three main subcomponents: development style, development architecture, and technical documentation. The software development approach was applied in a case study in cooperation with Ericsson. At the completion of the case study, questionnaires were administered to Ericsson employees to evaluate the success of the software development approach. The results showed that the quality of the software product was high and Ericsson was pleased with the result. The results indicated that the development approach helped the case study be successful in some of the researched areas. The end results suggest that the proposed software process has the potential to be successful in other projects of a similar type and structure.

Keywords

Software development process, agile, modular, technical documentation
Acknowledgements

I would like to thank Ericsson and all the people there for allowing me to work with them on the case study project. They provided a real life development project scenario that I could not have otherwise created. I would also like to thank Paul Johannesson for being my reviewer and for the feedback he gave and the design science insight he provided. I would also like to thank Rahim Rahmani for acting as my advisor. Lastly, I thank my wife and children for supporting me throughout the duration of the masters program.
# Table of Contents

1 **Introduction** .............................................................................................................. 1  
  1.1 Background .............................................................................................................. 1  
  1.2 Problem .................................................................................................................. 1  
  1.3 Research Question ................................................................................................. 2  
  1.4 Aim ......................................................................................................................... 2  
  1.5 Significance & Originality ...................................................................................... 2  
  1.6 Related Works ....................................................................................................... 2  

2 **Extended Background** .............................................................................................. 4  
  2.1 Challenges in Software Development ..................................................................... 4  
  2.2 Accommodating Change .......................................................................................... 5  
  2.3 Inefficiency in Software Development .................................................................. 5  
  2.4 Architecture ............................................................................................................ 5  
  2.5 Methodology .......................................................................................................... 6  
  2.6 Documentation ....................................................................................................... 6  
  2.7 Artifact Knowledge Base ....................................................................................... 6  
    2.7.1 Development Styles - Waterfall ....................................................................... 6  
    2.7.2 Development Styles - Agile .............................................................................. 8  
    2.7.3 Architectures - Monolithic Application Architecture .................................... 9  
    2.7.4 Architectures - Modular Application Architecture .......................................... 9  

3 **Method** ...................................................................................................................... 11  
  3.1 Research Method Alternatives .............................................................................. 11  
  3.2 Selected Research Method .................................................................................... 12  
    3.2.1 Strategies and Methods Used ......................................................................... 12  
    3.2.2 Practical Summary ......................................................................................... 12  
  3.3 Ethical Considerations ............................................................................................ 13  

4 **Results** ....................................................................................................................... 14  
  4.1 Artifact Requirements ........................................................................................... 14  
    4.1.1 Requirement #1: Development Style ............................................................. 14  
    4.1.2 Requirement #2: Architecture ...................................................................... 14  
    4.1.3 Requirement #3: Technical documentation .................................................. 14  
  4.2 Artifact Construction ............................................................................................. 15  
    4.2.1 Style ............................................................................................................... 15  
    4.2.2 Architecture ................................................................................................... 16  
    4.2.3 Documentation ............................................................................................... 17  
      4.2.3.1 Documentation Guidelines ...................................................................... 17
4.2.3.2 Location of documentation: .............................................................. 17
4.2.3.3 Meta-level documentation details: .................................................... 17
4.3 Demonstrate Artifact .............................................................................. 18
  4.3.1 Ericsson Background .......................................................................... 19
  4.3.2 Project Background ............................................................................. 19
  4.3.3 Project Execution ............................................................................... 20
4.4 Evaluate Artifact .................................................................................... 20
  4.4.1 First-hand perspective ......................................................................... 21
  4.4.2 Management Perspective ..................................................................... 22
  4.4.3 Technical Perspective .......................................................................... 23

5 Conclusion ..................................................................................................... 25
  5.1 Summary .................................................................................................. 25
  5.2 Ethical and Social Consequences ............................................................. 26
  5.3 Limitations ............................................................................................... 26
  5.4 Future Work .............................................................................................. 26

References ....................................................................................................... 27

Appendix ........................................................................................................... 28
  A - The agile manifesto principles (Beck, 2001) ............................................. 28
  B - Case study project description document: ............................................... 29
  C - Case study use case document ............................................................... 31
  D - Final software screen shots ................................................................. 33
  E - Management perspective questions and responses ................................... 41
  F - Technical perspective questions and responses ....................................... 43

List of Figures and Tables

Figure 1 Waterfall Method (Ruparelia, 2010) ..................................................... 7
Figure 2 Agile Model visualized (Benson, 2010) ................................................ 8
Figure 3 Research Strategies and Methods (Denscombe, 2003) ....................... 11
Figure 4 Example of what Meta-Level Documentation should look like ............ 18
Figure 5 Technical questions and results........................................................ 24
1 Introduction

1.1 Background

In a very competitive and ever increasingly technical society, the effectiveness of a company’s software development approach is vital to their success. I have experienced first hand the ineffectiveness of several different approaches, and have used that knowledge combined with theoretical research to propose a new combination approach for developing software. It is a three-part process that focuses on using modular architecture, agile methodology, and visual technical documentation. The main goal of the approach is achieve greater resiliency to change, and to be more capable of delivering software that meets the customers needs. In addition, the approach is an attempt to make the transition between teams and individuals more smoothly with the help of modular software and higher quality documentation.

The remainder of the research will be presented in the following manner. The first chapter describes the background area and explains the main purpose of the research. It also addresses the problems and challenges that exist and detail how it is related to existing research. Chapter two covers more about the specific issues the method will attempt to address and contains literature research about potential components of the method. Afterwards, chapter three explains the methods that were selected to conduct the research and case study and why they were selected over alternative methods. Chapter four goes through the core design science areas: requirements, construction, demonstration, and evaluation. Finally, chapter 5 provides a summary and final conclusions as well as discussion of issues, limitations, and future work in the research area.

1.2 Problem

Software development is a complex process that is often faced with changes in requirements, technology and people. The pure nature of the software development process makes it very susceptible to inefficiency. I have personally witnessed multiple different attempts from software companies to make the development process more efficient by selecting a specific approach to achieve better efficiency. Each attempt I have seen has fallen short in being able to minimize the loss of time and money throughout the software development process. My experience and the research conducted taught me about some of the causes to these problems. I believe there are three main areas of focus to determine a complete software development approach. The industry data and processes I have been exposed to have only properly focused on one area. By focusing on one area, the companies improved their process, but also left large gaps of inefficiency in their development approach. It was obvious that the processes used were not optimized to their highest potential, leaving room for improvement. The three main areas of focus that need to be addressed thoroughly are: the software architecture which affects how the software is constructed, the methodology which affects how the development process is carried out, and the software documentation which affects how the software is transitioned between people. The problem can be summarized as a lack of a comprehensive software development approach that can be used to gain greater efficiency in the execution of software development.
1.3 Research Question
Can an agile modular software development approach be efficient and capable of helping software projects meet requirements, handle change, and the transition of people?

1.4 Aim
The aim of this research project was to propose a software development approach that would be efficient in managing the challenges that are often present in the development process. In order to evaluate my proposal, I will empirically test my proposed approach to determine if my combination of modular architecture, agile methodology, and visual documentation can achieve greater efficiency, and to evaluate how my approach copes with changes in requirements, technology and people. The goal is to determine whether or not my proposed development approach should be recommended to companies, organizations and teams as a pattern for how they should conduct their development.

1.5 Significance & Originality
The results of this study will be relevant for companies and organizations that develop software. Some software companies may be forced to follow a certain methodology or architecture because of the type of software they are creating, but all others will be able to gain insight from the results of my research. Software companies will be able to see what aspects of my approach were effective and if any were less effective for each of the different components of my approach.

The software development industry is large and is continuing to grow. The ability to improve the efficiency of the processes used is of great value to the software community. As technology continues to develop, people want their software to do more and more. This means there will be a growing need for the software process to facilitate smoother expansion of existing software and more efficient development going forward. The results obtained will provide an indication of the cost of change when following this specific approach, which has been constructed in an attempt to handle the challenges described. It may be significant in suggesting a better way of executing the software development process to achieve better efficiency and maintainability.

The research is original in nature due to several key points. The overall originality stems from the unique combination of research areas that together contribute to the purposed method. In addition to the high-level combination of established areas, the sub component area of technical documentation is original. The technical documentation approach uses some practices that are known and established, but purposes a technical documentation technique and strategy that is completely original.

1.6 Related Works
The work of this thesis belongs within the department of computer and system sciences in the degree program “Engineering and Management of Information Systems”. It is related to engineering because of the software development aspect and related to IS management because of the project management aspect. The related works are vast and varied. As I dove deep into the research for this thesis I found that there has been large amount of work and research conducted into very specific areas of the different software development processes. There is also a considerable amount of research that has been done comparing different methodologies or researching moving a specific industry from one
methodology to another. What I did not find very much of was the higher-level combination approach that I was interested in researching. Much research has been conducted on the agile development style, but I found little research combining agile with other steps of the development process. The majority of related work was more granular than I wished to research. The inspiration to do the research at this level came from first hand experience with various pieces of the development process in industry. I found that even using a superior methodology with a strong research background wasn't enough. I recognized a need for a high-level development process in order to achieve greater overall success. This thesis is closely tied to previous research in the areas of style and architecture and relies on that previous work in incorporating those components into the proposed software development approach.
2 Extended Background

2.1 Challenges in Software Development

Software development is a vast and complex science that has many opportunities and challenges associated with it. In order to understand the current problem, one most understand the area of software development. The area of focus is high-level software application development. High-level software application development is the process of building software services or applications from one of many high-level programming languages. Popular high-level languages include but are not limited to: Java, C#, PHP, Python, and Perl. These languages are used to create software applications comprised of three main parts: data, interface, and logic. Applications and services are typically data centric; meaning the purpose of the applications is to manage data. An application may manage data by storing and retrieving customer data, providing access to a set of documents, or making calculations based on different data values. The data can be stored and used in a variety of ways with the most common being a database or file system. Another distinct feature of this type of software is that it always has some type of user interface. The user interface could be accessed via an Internet browser, or by launching an application on a computer, smart phone, tablet, or server. The interface allows the user of the software to interact with the software data. An email application interface could help a user create a new email as data, store that data and then send it to the recipient. A photo application could be used to open images and manipulate the image data and save, send, or print the image data. Logic is the brain of the application and it is comprised of all the code that does the majority of work for the application. The logic provides functionality, the data provides the content, and the interface allows access to use the application.

Understanding the basic foundation of this software allows us to move on to the processes, problems, and potential that is associated with executing the development of it. As Thompson says, "Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems" (Thompson, 1999). It is apparent when we look around at society today that software is a very important area for business. If simple processes can be improved and made more efficient, then surely larger complex processes like software development can also be optimized. The more efficient the software development process is, the greater the financial growth of the business will be. In a perfect world, we would have a perfect software process that would yield us the highest possible financial return for the software we create. However, we do not live in a perfect world, and there is no perfect software development process. There are currently many different approaches to software development and because there are many different aspects involved in a development approach, many combinations of approaches can be used in order to attempt to gain higher efficiency. Working as a software developer for the last seven years for three different companies has given me first hand experience with the challenges encountered with the software development process. There are many areas that may be the root of process problems. I will focus on the ones that have been most apparent to me.
### 2.2 Accommodating Change

One large challenge is change. Change occurs in many ways and is always part of software development. There may be change in requirements, change in needs, change in results, change in data, change in the interaction with the software, change in technology, change in development teams, change in direction, change in financial support, change in the market and much more (Tanabe 2008). Most of the changes that occur affect the software development process and the software must be modified as result of those changes. Change is inevitable and instead of avoiding change, the software development process should work to handle and accommodate change in the best way. Change can be accommodated best if it can be isolated to certain areas, and if it can be incremental and smaller in nature. The goal to combat change should include specific tactics in order to minimize the size and amount of change, and limit the area of software affected by the change. If the software process is not prepared to accommodate change, then it is highly likely that the software process will be negatively impacted by change that happens. Change is such a big factor with software development because it has a large impact on both the financial return and the finished software. All change is not bad, there may be changes in technology or the market that allow a business to grow at a much faster rate by modifying their software in a way that was either not feasible or did not make sense previously. Change is an expense. It costs money to make changes to the software because it entails a given amount of planning, design, and development hours to be spent in order to make the changes. If the financial improvement from software change is greater than the cost of the development, than it is valuable to complete the work. The more improvements that can be made that will lead to greater financial return and higher customer satisfaction the better. It is imperative then to select elements of the software development approach that will assist in making the cost of adaptation as small as possible. This will promote the decision makers in charge of the development to invest in updating and modifying the software more to increase their financial return and customer satisfaction. In turn, by continually updating the software in reaction to external factors the software will be less likely to fall far behind the current technology of the day.

### 2.3 Inefficiency in Software Development

There are certain aspects of a software development approach that are inefficient and they contribute to lowering the overall efficiency of the approach. These parts of the process may be overlooked altogether or not given enough attention. Making the final software product meet the client’s requirements is part of this challenge. A complete set of requirements is not always available from the beginning of software project and notwithstanding; requirements can change for a variety of reasons as described previously. The software development approach should help “enhance the quality and perfect software functionality” (Wang 2008) of the final product to satisfy the client’s needs in form and functionality.

Inefficiencies can also come from each of the three core pillars that make up this software development process. These three elements are architecture, methodology, and documentation.

### 2.4 Architecture

The architecture of the software approach is the portion that deals with specifically how the software is constructed. The programming language can have some impact on the architectural approach, but more importantly is the high level architectural design that comes into play for a given software
approach. There are many different types of architectures that can be used. In some cases, the programming language may be dictated by means other than the development approach, and its role may not actually have a positive or negative effect on the process. The architecture is what dictates how the software code is structured and how the different components of the software interact together. The architecture is the part of the process that specifies how the interface is connected to the logic and how the data is stored. The layout of these elements described in the first section of the background is a critically important aspect of architecture because it can affect the overall effectiveness of the process. The architecture is critical because it is the determining factor on how the coding is executed. Different architectures may vary greatly in the amount of code that is required to do a certain task, or the relationship between the different components.

2.5 Methodology

The methodology has more to deal with the actual process of carrying out the development and is less concerned about the actual technology used and way the technology is constructed. The methodology directs the management and workflow of the development. As with the other areas, there are many different approaches for managing and developing software. One main aspect of the methodology of development is ensuring that the end result fulfills the purpose for which it was created. When a specific set of goals has been determined, it may be possible to construct an approach that can be used to best meet those goals in terms of the software development process. There is great depth to this area, and entire sets of books have been written focused on only a single methodology approach. The methodology alone can be optimized and scrutinized to a large extent. The purpose of this research is to achieve a more holistic understanding of the software development approach, and therefore the specific methodology can only be a portion of the bigger picture.

2.6 Documentation

Software documentation is a prime example of an undervalued area of the software development approach. Documenting the code and writing other documentation about the software is a standard procedure that is part of the majority of software processes. However, from my experience the documentation that is completed is only helpful a small percentage of the time, while the rest of the time is spent in frustration trying to figure out why the code was constructed in a certain way and wondering where to locate a certain piece of functionality. Software documentation should be done in a way that helps other developers understand what they need to know and where they need to find it, and how it works. If it is done in an effective way, it can save precious time of the developer and make the overall process more efficient. If documentation is not present in the process, or if it is not being done in an effective way then it becomes a real detriment to entire process.

2.7 Artifact Knowledge Base

2.7.1 Development Styles - Waterfall

The waterfall is one of the oldest and most traditional styles of doing software development. The waterfall style “tends to be the standard against which other development approaches are compared” (Mitchell, 2009). It is easy to understand where the name came from, because it describes the way the
process flows down like a waterfall from step to step of the process phases. The waterfall method in regards to software engineering originated from an article written by Winston W. Royce in 1970, where he laid out the design of having seven different development phases that occur sequentially (Royce, 1970). The seven phases are: evaluation, requirements, analysis, design, development, validation, and deployment (Ruparelia, 2010). As Sedano mentions, the waterfall method is typically associated with large and complex development projects (Sedano, 2011). Some may argue this is an old and outdated method, and should not be considered a style option any longer. However, in a large survey to individuals collectively responsible for thousands of projects, it was reported “a lot of people identify it as their development method of choice” (Laplante, 2004).

The waterfall style is characterized by thorough planning and preparation at the beginning of the process. As can be seen in figure 1, steps 1-4 are all about preparation, and the development itself doesn’t actually start until step 5. Another characteristic of the waterfall style is the requirement of the creation of many types of documents associated with the process steps. The document creation can take a significant amount of time and it should be remembered that the process must completely finish the current phase before moving on to the next. A very important aspect of Royce’s revised model is the feedback and verification portion of each phase. At the end of each phase, the work from that phase is validated, tested, or verified to be certain it meets the expectation of the previous phases. The dotted line in figure 1 from Ruparelia, you can see an extra feedback loop that can accommodate problems encountered in design and development that need to return to requirements and design respectively (Ruparelia, 2010). The waterfall approach is a strict model in that it involves making big key decisions for the entire project upfront and then holding to the design and requirements decisions later in the development and testing phases. The documentation of the requirements, design, and implementation help dictate that the project goes according to the plan. The customer interaction takes place primarily at the beginning of the project when the decisions are made, and at the end when the product is delivered.

Figure 1 Waterfall Method (Ruparelia, 2010)
2.7.2 Development Styles - Agile

The agile development style “refers to a family of best-of-class iterative software development practices, methods, and techniques” (Vanderleest, 2009). It arose as an alternative to the classic waterfall method. It is rooted in an iterative style that strives to add functionality incrementally through many development iterations or cycles. In 2001, a group of developers met and together created what they called the “Manifesto for Agile Software Development” (Beck, 2001). The manifesto wasn’t necessarily the very beginning of what is referred to as agile development, but it did outline the main concepts, and principles of the methodology. The four core bullets of the manifesto are: “individuals and interactions, working software, customer collaboration, responding to change” (Beck, 2001). In addition to the main definition of agile development, the Agile Manifesto provides a list of fundamental principles that make up the agile methodology (Appendix A). The basic premise of these core values and principles is that there should be constant communication with the customer to allow the development process to evolve with time. The shorter cycles of development are a way of breaking down a large problem into much smaller problems. In addition, the shorter cycles help push the most important development to the top of the priority list. Agile development relies on constant communication and many iterations rather than high levels of planning up front with a lot of documentation.

As can be visualized in Figure 2, the agile approach starts with an overall strategy, with a vision and goals of the whole project. The project is then broken down into releases that have both a release plan and a backlog. The backlog is a set of features or functionality that could not be completed in the current iteration and must be prioritized for a later iteration or canceled. The most fundamental element is the iteration, which also has an iteration plan and review. A release consists of multiple iterations. The iteration consists of working days, with the goal to produce a working version of the software each day. Often there is a build created daily of the current version of the software. Each day work is completed on tasks that are part of the current iteration along with refactoring.

Figure 2 Agile Model visualized (Benson, 2010)
Agile development is built to handle change in requirements, and Laplante mentions that change in requirements is very typical in development projects. He said that “the more we understand something, the more we realize the flaws in our initial assumptions and conceptions” and then went on to say that if our development process is not adequately designed to handle change in requirements that the cost of doing so can “escalate exponentially” (Laplante, 2004). Agile development can help change be a driver for improving the quality of the product instead of being a cost burden. In addition, when comparing an iterative approach to a staged waterfall approach, the iterative style “has been widely accepted as definitively better” (Vanderleest, 2009).

2.7.3 Architectures - Monolithic Application Architecture

The type of architecture that will be referred to as monolithic is simply the organizing of the fundamental application elements together in a single component or unit. This can be explained as an integrated architecture approach as opposed to a modular separated approach. In this monolithic style, the user interface elements can be mixed with the program logic, and the data management code. It doesn’t mean that all elements will always be present in all software classes of the code, but the design allows for them to all mixed together into one unit. The monolithic name is being used for lack of a better term, to indicate that all different types of the foundational architectural elements of the application can be used together in one block.

The monolithic architecture style seems to be generally looked down upon when you read people’s comments and responses on forums and blogs on the topic. In his paper on the new age of software development, Aoyama classifies monolithic architecture as “conventional” and lists it along with older styles of the different steps of the software development process. He also counsels on how to avoid monolithic architecture and depicts it in a derogatory fashion by illustrating one big block with a dinosaur next to it (Aoyama, 1998). The positive aspects of a monolithic approach include a lower complexity of interaction between parts when multiple parts can be gathered into a single unit. Another positive is the ease of seeing a whole process in one place, for example you could see the user interface code along with the processing of data from the interface and the persisting of it to a database in a single class and file. The ability to use logic to manipulate the user interface, which is typically less dynamic, is another benefit of using a monolithic approach to the architecture.

2.7.4 Architectures - Modular Application Architecture

The word modular can convey different meanings across different technological areas, but for software applications it means that the different fundamental elements are separated from each other. The code for the user interface is kept separate from the program logic and the data persistence. It is essentially the antithesis of a monolithic approach. Modular application architecture also embraces the idea of grouping together common functionality and it making the functionality reusable by other sections of the application. The more modular the application is, the easier it is to reuse it in other situations. The use of modular style architectures has been done to help combat increasing complexity especially related to user interfaces (Peng, 2010). A common emphasis with modular architecture design is “the separation of functionality from presentation” (Peng, 2010). Peng also emphasizes that the separation of the interface from the functionality is one of the most valuable attributes of the development of modern software (Peng, 2010). Another potential benefit of using modular style architecture is that you can support multiple different types of user interfaces without changing the underlying functionality code. Using modular architecture can add complexity to the structure to separate out the different pieces of the application, but it can also lower the impact of change when modifying the application. If there is a need to change the functionality, it can be done without
affecting the interface code. Modular projects also provide organization for working with a large code base, which can make it easier to locate specific code.
3 Method

The research from this thesis is contained within the framework of design science (Johannesson, 2012). Design science covers a broad area of scientific research with different types of design science artifacts and knowledge types that can be produced to increase understanding or explore potential. The design science artifact that is being produced in this thesis is of the type: Method. The method that will be produced is an agile modular software development approach. Design science methods “define processes for how to solve problems and achieve goals” and “how to work in order to create artifacts” (Johannesson, 2012). The purpose of this research is to increase knowledge and there are many different types of knowledge that can be produced from design science research. The type of knowledge that is to be produced from this model is prescriptive knowledge. Prescriptive knowledge has two main sub components: prescriptive models and prescriptive methods. The prescriptive model is really the actual process or artifact method described above, and the prescriptive method is “a statement about the effect of using the model or method” (Johannesson, 2012).

3.1 Research Method Alternatives

There are many different types of methods and strategies that are used in conducting research projects. In Denscombe’s book directed at smaller scale research projects, he outlined the different methods and strategies that can be used to assist with social research.

<table>
<thead>
<tr>
<th>Strategies</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surveys</td>
<td>Questionnaires</td>
</tr>
<tr>
<td>Case studies</td>
<td>Interviews</td>
</tr>
<tr>
<td>Experiments</td>
<td>Observation</td>
</tr>
<tr>
<td>Ethnography</td>
<td>Documents</td>
</tr>
<tr>
<td>Phenomenology</td>
<td></td>
</tr>
<tr>
<td>Grounded theory</td>
<td></td>
</tr>
<tr>
<td>Mixed methods</td>
<td></td>
</tr>
<tr>
<td>Action research</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3 Research Strategies and Methods (Denscombe, 2003).*

Denscombe points out that there isn’t a specific pattern that you can follow to determine the best strategies and methods to use. Each strategy and method has pros and cons and each one is usually better at fulfilling a specific need (Denscombe, 2003). After reviewing all of the strategies it became clear that ethnography, phenomenology, grounded theory and mixed methods were generally irrelevant for the purposes of this research. Action research was a possibility, but it is focused on less general results from practical work. Experiments could have been used, but they would be less effective than a case study in obtaining more holistic results. Surveys could have been used, however,
the results would be lacking in real application specifics and it wouldn’t provide a holistic view of the combination of components. It became clear that a case study would be a solid option to test the method in a real world application. The case study also provided a way to test all of the components together to give an idea of the holistic success of the method, and determine if any of the sub components affected each other. The use of methods was less obvious, as both interviews and questionnaires could be used. Interviews were discarded as an option because they would be used to collect the same data as questionnaires and would provide less objective data results because the answers would reflect directly back on the interviewer. The observation method was not relevant as the individual doing the observing and the one being observed was the same for this case.

3.2 Selected Research Method

3.2.1 Strategies and Methods Used

In the actual carrying out of the research some additional design science principles were applied. The research will use document studies and questionnaires to gather data in addition to using a case study. As described in “A Design Science Primer”, the case study is important in helping “to paint a rich picture of a single object or situation to be used as a basis for obtaining a deep and comprehensive understanding of some general phenomenon” (Johannesson, 2012). The selection of these methods and strategies makes sense in several ways to achieve the purposes of the research. One way the case study can be used is in a theory led experiment to test a specific set of factors (Denscombe, 2004). This is exactly what I hoped to obtain from the case study. The hope was to gain data about the success of the proposed approach. A case study was a good fit for this research because it values the “holistic view rather than isolated factors”, “Natural settings rather than artificial situations” and “Multiple sources rather than one research method” (Denscombe, 2004). It was important to test the whole approach together, to test it in a real environment, and to be able to use both document studies and questionnaires to help support the research. Questionnaires were selected instead of interviews because they are more objective in this instance, and because nearly half of the questions relate to data that needs to be ranked on a scale. The scale questions as well as the data collection are more efficiently done with questionnaires. The documents method was selected because the case study would produce a number of documents in the form of code and documentation. For the research to be most effective those documents needed to be reviewed and evaluated. Using a case study together with document studies and questionnaires created a combination of strategies and methods that helped fulfill the aim of the research. The practical information about how the research was performed is described below.

3.2.2 Practical Summary

The software development approach has been researched and will be explained and argued from a theoretical standpoint. Each of the three main areas of the process will be explained and detailed. After the theoretically presentation of the approach, the empirical work will be presented. The empirical work consists of a case study test of the development approach done in cooperation with the radio department of Ericsson. The case study work at Ericsson will consisted of going through an entire development process in order to test each phase of the proposed approach. The project that was used for the case study was the creation of an application for Ericsson to interact with their software database configuration file that they use for their Radio Base Stations (RBS). The case study project required me to go through the planning, development, and delivery phases of a software project. I had
to gather requirements, schedule the work, conduct reviews, give demonstrations, and deliver the final product. The effectiveness of the development approach has been measured in a number of ways. One way is a social approach of my own response to the approach execution of the case study in comparison to the other different approaches that I have experienced. The empirical work has also been measured by comparing the established initial set of requirements with the final results that were delivered. Data has been captured regarding significant change encountered during the project duration, and will be used to scientifically illustrate the results of certain parts of the approach. In addition, questionnaires have been administered to several Ericsson employees to gain additional perspective into the effectiveness of the approach. The Ericsson development team member that is responsible for maintaining the case study project after its completion was questioned to assess the level of difficulty for him to take over the project and familiarize himself with the code base as well as his assessment of the technical documentation. After combining all the data, I was able to analyze the results and make conclusions based upon it.

3.3 Ethical Considerations

The ethical concerns that needed to be planned for and dealt with properly are those that dealt with the privacy of the intellectual property and proprietary knowledge that belongs to Ericsson. There is a certain amount of knowledge that was required to carry out the case study and that knowledge may not be shared publicly outside of the Ericsson network. In addition, the full implementation details of the case study including the code base and affiliated data set is proprietary information that may not be shared as part of the thesis report. To ensure that proper ethical conduct is performed, several steps have been taken. Privacy agreements were signed at Ericsson, and an Ericsson computer was allocated for use for the case study. All Ericsson sensitive data was kept on the Ericsson computer, which was equipped with data encryption software. The information related to the case study was reviewed and authorized before the completion of this thesis. In addition, Ericsson employees who participated in the questionnaires were informed that the data they submitted would be included as part of the thesis report.
4 Results

4.1 Artifact Requirements

The artifact that is to be constructed is a software development approach. It is a way of conducting and executing the development of software so that the development process can handle change adequately as well as other factors that face a modern development team. The software development approach or process is of the type: method. If the method is successful and used properly it should be an efficient way of working through the process of design, requirements, development and delivery of software. This method to be created is a unique combination of components with more extensive history and research. The purpose of this section is to describe the different requirements that need to be met for the construction of the proposed method. There are three main requirements that need to be fulfilled to make up three method components. These three components will be used to build the software development approach.

4.1.1 Requirement #1: Development Style

There are many different types of development process styles that have been conceived, tested, and used. Many of the styles have been developed as result of the lack of certain attributes in existing styles. In order to make a selection of styles for use in this method, the focus should be aimed at the high-level categories of most used and popular styles. The development style primarily dictates how the management of the project proceeds in terms of executing the design, gathering the requirements, and conducting the development.

4.1.2 Requirement #2: Architecture

The word architecture is quite ambiguous and can have many different meanings even within a specific category of knowledge. The Merriam-Webster dictionary has five definitions listed with the fifth stating that architecture is “the manner in which the components of a computer or computer system are organized and integrated” (Anon, 2012). Even by that definition, it could be interpreted in many different ways for a variety of different circumstances. The architecture that is relevant for this method is the architecture of the software application, meaning the way in which the software application components are constructed. The software application contains several fundamental elements such as a user interface, program logic, and data management. The way in which these elements are constructed and designed to interact with each other is the architecture that this research is concerned with.

4.1.3 Requirement #3: Technical documentation

Technical documentation is a very critical factor in being able to adequately deal with change and properly execute the different phases that occur in the software development process. Documentation is the special piece of practicality that works in conjunction with the style and architecture to make a complete development approach. It is an important part of what should be delivered with any software development project, but Bayer points out “in practice, however, the creation of appropriate documentation of software is largely neglected” (Bayer, 2006). The documentation requirement has been included because with development there are almost always multiple developers working on the
same code base. The changes that occur in both code and developers can be greatly hampered by the lack of effective and helpful documentation. The presence of the right kind of documentation can facilitate a more effective and smooth development process while the lack of documentation can become a stumbling block for the developers who are trying to complete the work. There is a fine line that must be walked in regards to documentation. If there is a requirement for too much documentation, it then becomes more of a burden on those required to produce it than a benefit for those who will use it. It can also be a burden to update and maintain large amounts of documentation. On the other hand, if there is too little documentation or the wrong kind of documentation, then the developer is left in confusion to try and understand the code and structure by himself. As Lethbridge says, “most software documentation is not updated consistently. We must find powerful yet simple documentation strategies and formats that software engineers will likely maintain” (Lethbridge, 2003). Later, Lethridge also counsels that in order “to achieve greater documentation relevance, we need to find ways to increase its power, simplicity, or preferably both” (Lethbridge, 2003). Efficient and helpful documentation done right, should not be a heavy burden on the creators, and should be a great help in clearing the path to understanding for the people who will refer to it.

4.2 Artifact Construction

The proposed design science artifact is a method for doing software development. It is unique in many ways, but especially for its special combination of style, architecture and documentation selection. Much research has been conducted on a very granular level of the different styles and architectures that were examined. This high-level combination approach is both new and exciting. It combines some well-researched foundational pieces with a new perspective on documentation to create a very practical and effective development approach. So much of current research focuses on such small aspects of the software development process, but this approach abstracts out some of the more specific details in order to produce a fully functional method for doing high quality software development. This unique approach will be described by detailing the specific main components that were selected and why.

4.2.1 Style

The high-level design style that has been selected for the construction of this software development approach is the agile development style. The largest motivating factor for selecting agile as the style to use is because of its emphasis and design to be able to accommodate change in the development process. Laplante agrees as he points out that “the modern reality of software development is that change is unavoidable and must therefore be explicitly accommodated in the life cycle. It is not an error that must be fixed; it’s a natural aspect of system construction” (Laplante, 2004). Both in the design science research conducted and in the practical work experience with software development, it has been clear that focusing on change is extremely important. The iterative agile style is conducive to change in the different aspects of the development cycles. Agile methodology breaks down the project into planned iterations that encourage learning throughout the course of the development project. From personal experience, it has become evident that even large established companies do not have all the required knowledge to complete all the planning and requirements up front as the waterfall and some other styles dictate. On the contrary, the iterative agile approach becomes a catalyst for improving the design and requirements of the project along its development. Companies are also given the chance to both learn about the capabilities of current technologies and how they can affect and improve their project as well as take advantage of emerging technologies that arise during
One potential drawback with selecting the agile style is that it has a dependency for constant communication throughout a project. Ruparelia points out that this can be a difficult task for some large projects to achieve (Ruparelia, 2010). In order to use the proposed software development approach, it is important that the proper organizational infrastructure is in place to allow the necessary communication channels to be in place.

### 4.2.2 Architecture

The architecture used in this software development approach is modular architecture. In a continued effort to meet the initial goals of the method, the modular approach to architecture is better suited to accommodating different types of change. Two ways that modular architecture is a better solution for handling change over a monolithic approach are organization and separation. The organization of the code base for a project done using a modular architecture is much easier for developers to locate specific sections of code. The code organization also helps by allowing different types of employees to interact on the code together with less conflict between them. A typical example is when a back-end developer and a front-end designer are working on the same module. The modular architecture dictates that the user interface code is separated from the processing logic that is associated with it. That allows a front-end designer to easily modify the interface code separately from the back-end developer who is writing code to process information from the interface. The same organizational characteristics are relevant for a group of back-end developers working simultaneously. They could each be working on different modules of the same project and have little to no conflict as they work. The separation of the code into different modules and into different types of code such as interface or backend assist with handling change because it isolates code functionality such that major changes can be made to one portion of the application without affecting the rest of the application. A user interface could be completely changed leaving the backend logic untouched. The separation of code from interface to logic and from module to module may require some additional overhead in the beginning, but more than makes up for itself as a project grows in size and complexity. In addition to change, modular architecture supports the agile way of building a working application and then iteratively improving it and adding functionality to it. New modules can be added as more functionality is added, and existing modules can be modified to adjust to changing requirements and functionality making a much smaller impact than would occur using a non-modular approach. Managing complexity is yet another reason for selecting a modular architecture over other types of architecture. As technology improves there is an increase in complexity, the added complexity caused the creation of modular types of architectures in order to keep the development process running as smoothly as possible (Peng, 2010). When evaluating the decision from an academic research perspective it became quite clear that modular architecture has more advantages than non-modular approaches. Furthermore, for this proposed software development approach, it fits nicely with and complements the agile style selected. From my practical work experience in the field, I have been alarmed to find that modular architecture has not been used as much as I would have expected. I expect that the number of use cases for using a non-modular architecture would be a small percentage of the overall number of projects being completed. The decision to use a non-modular architecture when a modular would have been a better fit for the project is not always the result of a conscious decision. In a couple of the cases I witnessed, the non-modular architecture was used due to the lack of an existing modular framework to enable the modular style to be more easily adapted or the lack of a decision altogether with the non-modular style being the default option. To attribute such little significance to the architecture design is a gross failure in perspective. The modular architecture has been selected because of the attributes it has, and the structure it helps give to the development project. Used properly, it should help make the development process smoother and more effective than it would have been otherwise.
4.2.3 Documentation

The documentation portion of the software development approach is a unique hybrid approach to doing technical documentation. It borrows some common practices from within technical documentation, but also takes a new angle on other aspects. It is designed to be lightweight with limited burden on the developer, while at the same time providing key insight for those who need to refer to it for guidance and understanding. Technical documentation can be done in a variety of ways, but no matter which way it is done there are some core principles behind how it is done that applies to whichever type or style is used. It is very important that the message of the documentation is clear. The comments are “not to impress, but to inform with clarity” (Rivera, 2004). The documentation should be concise so that is both easy to read and easy to maintain. One of the major problems with documentation is maintenance and it is one of the main reasons why it should be clear and concise (Lethbridge, 2003).

4.2.3.1 Documentation Guidelines:

Each place that documentation is required, there should be a clear and concise comments that describes the how and what the code is designed to do.

4.2.3.2 Location of documentation:

- All class definitions
- All non-trivial method declarations
- Complex logic sections within methods
- Meta-level overview

4.2.3.3 Meta-level documentation details:

The meta-level documentation is responsible for visually showing important relationships between modules and classes, class hierarchy inheritance structures, and for giving module level comments about purpose and function. It is important to note that not all relationships need to be modeled. Trivial and highly intuitive relationships may be left un-modeled to leave the documentation clean and uncluttered. If a given module is too large to have the meta-level documentation fit on a single page, then the module documentation should be logically divided so that similar functionality is grouped together with each sub group fitting on a single page. The visually represented relationships can be done using standard UML. Classes should be modeled simply using class names. When classes are of a special type such as Abstract or Interface, the type should be included, but otherwise full class details do not need to be present. The meta-level documentation is intended to give a high level understanding of how the project is constructed and should not be concerned about specifying low level details. As you can see in the example in Figure 4, all of the color classes inherit from the base color class. In this example the color classes themselves are trivial and are not shown on the diagram. Instead, a short explanation is given with the square brackets to indicate that all other color classes inherit from the base.
4.3 Demonstrate Artifact

As it was discussed in the method chapter, a case study was determined to be an adequate and appropriate way to test the software development approach that has been proposed. A case study is good because it tests the artifact in specific circumstances and allows you to gather information about the potential success of the general phenomenon (Johannesson, 2012). A case study is a real world testing of the artifact in a specific set of circumstances. In following the guidelines outlined by Denscombe, it is required to first select which type of case you will conduct: typical, extreme, test-site for theory or least likely. It made the most sense for this research to select the typical type because the goal is to understand how effective and efficient the method is for general cases of similar circumstances (Denscombe, 2003). It may be argued that it is not possible to make generalizations from a single case study, but as Denscombe points out there are a few arguments to the contrary. The first is that while a case study is specific in nature, it is also a “single example of a broader class of things”. If something is true for the specific thing, it may be true for the type that it belongs to as well. The second and third points are similar and they say that if there is enough detail about the case study, then the relevance of it for other studies should be based upon how similar other cases are that are related to it (Denscombe, 2003). The other checklist items that I ensured were followed in carrying out the case study include the following: natural occurring situation, self contained case, case boundaries, understanding of generalizations, use of multiple methods, and holistic perspective (Denscombe, 2003). A case study doesn’t necessarily provide absolute results in terms of success or failure. It does provide a real world trial that can illustrate strengths and weaknesses of an artifact. If a case study
succeeds or fails during the execution of the method, it can provide some insight into the probability of success or failure for similar types of projects. There was a development project identified at Ericsson that suited well as a case study for this thesis.

4.3.1 Ericsson Background

Ericsson is a large Swedish company that was founded in 1876. It originally started as a Telegraph repair shop but has since grown into one of the largest telecommunication and data communication providers in the world. One of the biggest parts of their business is to design, build and deliver the equipment necessary to create mobile phone networks. Their customers consist of large telecom corporations from around the world (Ericsson-History, 2012). Ericsson has many different departments and divisions to complete the complex engineering required to produce large-scale mobile networks. One of those departments is the radio department, which works on radio units, which are part of Radio Base Stations (RBS). The RBS units are used as part of mobile networks that allow customers to access the network via radio wave connections to the RBS. It is important to understand that the radio team is responsible for a wide variety of radio hardware products. They must create drivers and configurations and everything else needed to make the radio hardware equipment functional as they deal with continually evolving technology and the changing of electronic boards and configurations.

4.3.2 Project Background

The radio department had a problem that needed a software solution. The radio department is responsible for keeping track of large sets of hardware configurations for all of the different platforms and hardware variations that they have produced and are working on producing. The large set of configuration parameters that exists has not been maintained in very systematic and organized way. Initially there was not very much variation in the radio hardware and it was very simple to keep track of the configurations in a single text file. However, as time went on, the number of variations began to grow rapidly until it reached its current state. The format for storing the data has changed very little, but the number of configurations and people accessing it has increased dramatically. Having the set of configurations in a large text file caused many problems. It was difficult to make modifications to existing parameters and it was difficult to look up parameters in the file. It was not a user-friendly format for those unfamiliar with the file or configurations. Thinking about the future growth of the data and the existing state of the situation caused the radio team to identify this as an area needing improvement through software. The radio team decided they needed an interface for interacting with the large set of configurations, which is also referred to as the “software database”. The initial requirements were that the interface should to facilitate the basic CRUD (Create Retrieve Update Delete) functionality of the parameters and that the application should be able to output a properly formatted file that could be run through the compiler for use on the actual radio hardware. The parsing and loading of the existing data from the large file to a relational database was the first item that needed to be completed. The project plan document contains these initial requirements, schedule and more and can be viewed in Appendix B.

With the basic requirements established, it was time to select the technology to be used to carry out the development project using the proposed software development approach. The need to give many people access from several different departments of Ericsson was part of the motivation for choosing to build the software in the form of a web application. After researching the most popular web application programming languages and gathering feedback from some industry professionals, it was decided to complete the project using the Python programming language with the assistance of the
Django framework. The code would be written in the Eclipse IDE (Integrated Development Environment), and source controlled using Git, which is standard at Ericsson. The language and tools selected are platform independent and the Ericsson computer made available for the project was a Windows based laptop.

4.3.3 Project Execution

Following the guidelines of the proposed development approach the schedule of the project was roughly outlined to be eight weeks of development with four iterations two weeks in length. As you can see in Appendix B, the iterations were loosely planned and the rough requirements for established from the start. It was planned to first build out the core base functionality and then from there work through as many user cases as the time allowed making sure to prioritize the use cases to achieve the most crucial functionality first. The use cases can be seen in more detail in Appendix C. There were meetings scheduled to review the functionality built, determine use cases, and prioritize the use cases for development. As this project was being used for this thesis, I took on all roles of the development team myself. I was the analyst working with the client to gather requirements, determine the functionality and priority. I was also the development manager making ensuring the proper pieces of functionality were being completed with in the correct development periods. Lastly, I was the sole developer for the project responsible for writing and testing all the code to meet the requirements and use cases. The project development moved through the process as planned except for changing the completion date by about a week to accommodate difficulties in scheduling meetings and taking into account the Easter holiday break. Chronologically the projects main development followed as such:

- Parsing existing configuration file
- Core functionality: user authentication, basic CRUD operations
- Use case #1
- Use case #2
- Exporting all data to file
- Use case #3
- Clean up / tweaks / formatting / bug fixes

All of the use cases that were planned and documented were completed, demonstrated and approved by the Ericsson radio team. The functionality was all demonstrated to the Ericsson radio team throughout the project. The screen shots of the actual application were included in the final presentation to Ericsson as the completion of the project and can be see in Appendix D. There was also communication throughout the duration of the project both in person and electronically. All of the requirements and use cases were documented and verified to ensure accurate documents were present for development to follow. The continued work and development of the project were also discussed in the final presentation with the relevant Ericsson personnel.

4.4 Evaluate Artifact

The evaluation of the artifact was completed according to the research strategies and methods described in the method chapter. The use of document studies and questionnaires were used together with a three perspective strategy to evaluate the artifact. The first perspective was a first-hand view of the artifact and its elements. The second and third perspectives originated from Ericsson employees.
who were involved with the case study. The second perspective provided insight from a management viewpoint and the third perspective came from a technical angle.

4.4.1 First-hand perspective

The agile style was very effective from a practical standpoint in terms of planning and design. The decision making team at Ericsson did not know all the functionality they wished to be part of the software application at the beginning of the development process. They had an understanding of the basic idea of what the software application would be able to do, but had not made decisions about all the specific functionality details. The fact that those design decisions had not been made upfront was not a problem with the project being completed in an agile style. The agile style allowed the project to get started with very little overhead in terms of planning and design. In the planning meetings for the first iteration of development, we were able to make an iteration plan than consisted of the foundational underlying requirements for the software application. As the project continued on to the second and subsequent iterations, the requirements and functionality were conceived and planned. I believe that getting a simple working version of the software up helps open the vision of those decision-making individuals involved. The agile style was executed very smoothly, at each review meeting the functionality was shown and any changes were noted and added to the backlog, which ended up being completed in the next subsequent iteration. Towards the end of the project, the decision making team from Ericsson concluded that enough functionality had been completed to make a version one of the software. It was decided that all of the new ideas for use cases and functionality would be reserved for version two or later of the software. From the human perspective, the agile style did not do anything to lower the efficiency of the software development process, but instead was successful in providing sufficient planning and guidance with a small percentage of time in overhead. The agile process for this project was able to using on average, 5% of the work time for each week to conduct the planning, reviewing and other meetings. The development team consisted of only one member, which means for the case study the meeting time percentage was lower than it would most likely be for teams with a size greater than one. The project encountered less change than anticipated and the constant contact and short iterations made finding the problems and deciding upon the solutions very smooth. There were multiple positive impacts observed from using the agile style, and there were no negative impacts observed.

The modular architecture of the method proved to be very convenient in organizing, structuring and giving order to the source code. It was also effective in separating the program logic from the user interface code. That separation made it very nice to be able to initially create a very simple interface and then focus on the back end program logic and later come back and build a more sophisticated interface. If this had been a larger development team it would have been completely within the bounds of the modular architecture and the framework to build the interface independently. The framework that was used was the Django framework for Python. The framework itself was responsible for compensating for one of the weakness associated with modular architecture. The research on modular architecture showed that using a modular style could add complexity upfront, however, the Django framework included some creation scripts that created the structure for the application and each of the modules as well. It turned out that the use of the modular style architecture lowered the complexity in the long run. The code was clean and well organized, and there were fewer lines of code written than if I had used a non-modular approach. One main point of focus with selecting the modular architecture was the advantage it would provide in handling change. The amount of change that occurred during the Ericsson case study was trivial and didn’t provide us with as good of a test as expected. However, after analyzing the software commits to the repository for the project, I observed that each commit
generally contained only changes to files that were part of the module being worked on. The isolation of work meant that changes to one module could be done without being forced to modify the code for another module. The modular structure made it easy to start out with simple versions of each module and develop each of them out with more functionality throughout the development phase.

Some developers feel that technical documentation is a burden and that it slows them down and is a painful process. The objective of the documentation segment of the development process had a clear aim to be effective in communicating important information to the relevant individuals who would read it, while at the same time be easy for the developer to complete. It was very important that the burden of documentation was not too heavy for the developer; otherwise the developer might be tempted to not fully complete the documentation and might resent the process and produce lower quality work. As the developer on the case study project, I was responsible for all of the documentation of the process. After completing the documentation, I can validate that the process was not a burden and was not overbearing. The majority of the documentation that is laced throughout the code was easy to produce while writing the code. The most important thing for developers to do is to get in the habit of writing the documentation at the same time as they write a new piece of code. The meta-level documentation required extra time outside of the normal development time, but it was only approximately 2% of the total time spent working on the code, design, planning and reviews. The documentation in particular is most importantly reviewed by individuals unfamiliar with it and will be inspected more comprehensively in the technical perspective project evaluation.

4.4.2 Management Perspective

Jonas Löfgren is one of the project managers at Ericsson that was involved with case study project. He was responsible for managing the project from Ericsson’s point of view. He made sure the right people were involved in the meetings and presentations and helped with the planning, requirements and general project details. The aspects of the proposed software development approach that was evaluated from the management perspective are the style and the meta-level documentation. The architecture is relevant from the technical perspective as is the full evaluation of the documentation. The agile style dictates the process of planning and gathering requirements as well as the practical elements involved in completing a project. Jonas indicated that the requirement gathering proceeded smoothly and that “the needed functionality was implemented” (Appendix E) accordingly. Communication is a big factor when it comes to using the agile style and Jonas noted that it is vital in gathering requirements, planning, and presentations and that it is necessary to have communication outside those standard meetings and interaction points. The project was successful in communicating effectively in the standard meetings and in an ad-hoc manner with the proper technical individuals involved. The one limitation that Jonas observed was that the nature of the size and scope of the project makes it difficult to fully evaluate the style methods used in the development process.

The bulk of the technical documentation resides in the source code in the individual files of code. The meta-level documentation is stored at the top-level structure for each module and application. While the documentation is primarily technical in nature, for evaluation purposes, Jonas reviewed the meta-level documentation and provided feedback accordingly. He felt that the documentation gave 8 out of 10 level of understanding on the construction of the project, but in his opinion he would have liked to see some documentation detailing the setup of the project too.

Statistically Jonas rated the project 90% on fulfilling the requirement, efficiently completely the functionality within the time allowed, and in overall satisfaction with the results of the project. In general, he thought the project went very well, and was impressed with what was accomplished.
4.4.3 Technical Perspective

Albert Dunberg of Ericsson is responsible for the technical perspective of the evaluation of the case study. Albert was involved with the project and was the primary contact person regarding any technical details. He was also the most familiar with the use cases and requirements that were developed. As part of the evaluation of the project, Albert reviewed the source code and documentation that were produced and responded to a series of questions about the project. The feedback from Albert is the very valuable in determining the effectiveness of the proposed software development approach because Albert reviewed all of the code and documentation and he was involved in all aspects of the planning, designing and reviewing processes. Albert reviewed the code and documentation and because he had not seen it prior to the project’s completion he was able to give objective feedback regarding transition of the code to another developer. His feedback provides insight into how effective the chosen architecture and documentation actually was in helping a new developer to be able to look at the project and understand its construction and begin working on it.

The success or failure of the agile development style can be seen in part by the way the requirements, communication, and final results turned out. One part of the agile style that was covered in the research was the lack of major planning upfront like in the Waterfall method. This point proved to be a benefit in the Ericsson case study. Albert noted that Ericsson was somewhat “ill-prepared” in the beginning, but because of the iterative process, the team was able to narrow down the functionality to a select set of use cases that could be completed during each of the iterations. This concept correlates strongly with what I have seen from years of software development experience. Many times customers have many ideas but are unprepared upfront to make all of the planning decisions for the entire project like is needed with a Waterfall style approach. The lack of major planning upfront with agile development requires that communication is active throughout the project to ensure that all aspects of the project are understood correctly from both the development and management departments. Albert understood this principle clearly when he responded that “communication was central in the success of the project” (Appendix F).

The technical perspective on the documentation portion of the method revealed very positive results. After reviewing the code and documentation, Albert responded that the code was “well structured into modules reflecting the use cases” (Appendix F) that were created from the planning and requirement meetings. The structure of the code with the addition of the documentation enabled Albert to easily move through the code and inspect the important parts despite the fact that he had “next to no previous experience in Python” (Appendix F), the language that was used for the project. One big question regarding the documentation was whether or not it would be helpful in giving proper insight into the project for someone new to the code base. After rating his knowledge of Python and Django very low, Albert went on to rate his ability to understand the code, the structure, and ability to find a specific portion of code very high.

<table>
<thead>
<tr>
<th>Technical Specific Questions</th>
<th>1 (Worst) to 10 (Best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python knowledge</td>
<td>2</td>
</tr>
<tr>
<td>Django knowledge</td>
<td>1</td>
</tr>
<tr>
<td>Ability to make the structure understandable</td>
<td>8</td>
</tr>
<tr>
<td>Ability to make the code understandable</td>
<td>8</td>
</tr>
<tr>
<td>Ability to locate specific code</td>
<td>9</td>
</tr>
</tbody>
</table>
As can be seen in figure 5, there was an average of 8.6 out of 10 for all questions regarding the ability to use, learn, work with, and transition to the code base. The high percentage is a result of the development architecture and technical documentation style.

In conclusion, Albert expressed his opinion of the project in fulfilling the technical requirements with 9 out of 10, and his satisfaction with the final results of the project as a maximum 10 out 10. Understanding the limitations that accompany case study projects helps us understand that these results cannot be used to make a conclusive assessment as whether this approach would work in all other scenarios. The results show that from this case study, there were no drawbacks with any of the components used to construct the software development approach. It also showed that it was able to successfully handle many of the researched scenarios that were used to make the component selections. It was generally very successful, but all of the focus points were not fully tested due to unpredictable natural events. It is reasonable to generalize that because it was successful in this “typical” case, that other projects that meet the criteria to be classified in the same way might also be successful. To see all of the questions and answers from the technical perspective see Appendix F, and for the management perspective see Appendix E.

| Assume responsibility as developer for project | 8 |
| Ease of use with Model View Control structure | 10 |

Figure 5 Technical questions and results
5 Conclusion

5.1 Summary

Software development is an integral part of business in today's economy. The number of businesses and organizations that are becoming dependent on software is continuing to increase as more and more software is produced to make business more efficient. The need to create high quality flexible software quickly and efficiently prompted me to focus my efforts on the software development process. This thesis has presented the research and results of creating a new design science method for developing software. The method was summarized as an agile modular software development process with visual documentation. The method was constructed from three primary pillars of foundation: style, architecture, and documentation. The style is associated with how the project is planned, designed, and managed. The style that was chosen is the agile development style that is characterized by its iterative style and high communication over intense upfront planning and documentation. The architecture determines the way in which the actual software code is built and organized. A modular architecture was selected because of its ability to organize the code and encourage a more streamlined approach to iterative development and change in technology, requirements and engineers. The documentation that is part of the approach is a combination of best practice code commenting techniques and a hybrid custom approach for a higher-level perspective called meta-level documentation. The meta-level documentation leverages commonly used UML modeling style to create a flexible lightweight overview of important relationships of modules and classes. Together they create a loose high-level approach towards executing the software development.

Much of the research within this area is focused on a very granular level and there is less research focused on high-level combinations. The proposed development method may seem obvious in some ways, but the design decisions were influenced from more than seven years of software development experience where the most effective methodologies were not actually used. The practical truths I have learned from working in industry, is that companies often use pieces of effective and efficient development practices but do not actually use complete combinations of effective techniques. Each development environment that I have been involved with has had large gaps in efficiency due to the software process. In order to gain some understanding into the effectiveness of the development method, a case study was put into place with Ericsson to execute the method in a real scenario. The challenge with such a case study is that while you can detect flaws in processes and can discover potential and success it is difficult to make a conclusive evaluation about a given research field as a whole. Having said that, the approach was found to be effective within the constraints of the project. It assisted in dealing with a lack of upfront planning from Ericsson and it succeeded without extensive documentation and planning with short focused planning meetings and constant communication throughout the project. The focus on change was unable to be tested fully in terms of change in requirements and technology, but the documentation and architecture proved to be effective in dealing with human change. Software projects often have to deal with a change in development teams and in developers, and those changes can be costly if it is difficult to transition between developers. The case study showed positive results in the ability for a new developer to quickly understand the project code base. In general, the feedback from the case study was positive in favor of the components of the
development method, but it is suggested that further or more exhaustive testing and research is completed before more conclusive statements about the success of the method are made.

5.2 Ethical and Social Consequences

The ethical concerns that were identified in the method chapter did not cause any problems with gathering of valuable data from the case study. The data that was of real value to the evaluation did not fall into the category of data that should not be exposed to people outside of Ericsson. All ethical considerations were followed and there were neither positive nor negative implications as a result of the initial concerns.

The social consequences are difficult to measure within the framework of this thesis. It is reasonable to argue that a part of the future work could be to inform the software community about the software development method and results and gather feedback about the social impact with their respective development teams. Another angle the thesis could have taken, would have been to conduct many surveys to gauge the social response to the concepts contained within the development method. This was not done, as it has been argued that the most valuable data would be gathered from conducting a case study.

5.3 Limitations

The limitations are primarily those regarding the use of a single case study. It can be difficult to make generalizations except for those cases where the case clearly belongs to a certain type of situation or where the scenario relates closely enough to other cases to draw conclusions. The other limitations are that because of the scope and time of the project, the project itself is small in nature and would be more revealing if it was a longer development project with a larger development team. Another potential limitation is that because of the inability to quantify the result data with statistical data, the feedback from the developer and Ericsson employees is subjective because it comes from human beings, each of whom has a different perspective.

5.4 Future Work

The results of this study revealed mostly positive results. One case study is not conclusive, and the proposed future work would be to conduct broader testing of the software development approach to achieve a more comprehensive evaluation of the method as a whole and to do a more thorough investigation of the method to uncover any potential weakness and seek to improve them. Another point of future work relates to the general area of documentation. Due to time constraints and resources, the documentation portion dealt only with the technical documentation. Documentation is a large and important aspect of the software development process, and there is room for improvement in documentation areas outside of technical documentation.
References


Lethbridge, T.C. et al., 2003. Use Documentation: The State of the Practice Documentation:


Appendix

A - The agile manifesto principles (Beck, 2001)

Agile Manifesto Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
**B - Case study project description document:**

**Ericsson Development Project Overview**

The project is to build a web application that will include the following core features:

- Provide a graphical interface for Creating, Removing, Updating, and Deleting configuration parameters.
- Provide some level of validation for the creation of new parameters.
- Exporting of all parameters in the form of a text file.
- Provide a relational database and schema for storing the configuration parameters.
- Authentication to access the web application.
- Visual Documentation of the architecture layout and design.

**Preliminary Work:**

As part of the preparation for the building of the web application, the existing “software database” configuration parameters will need to be parsed and loaded into the new relational database tables from the text file they are currently in.

**Development Process**

The development period will consist of 4 development sprints. Each of the 4 sprints will be 2 weeks long. There should be a meeting at the beginning of each sprint where a priority list of the work for that sprint is decided upon, and the work from the previous sprint is reviewed.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objectives</th>
<th>Time Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 1</td>
<td>Core Features/User Stories</td>
<td>2 Weeks</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>User Stories and changes</td>
<td>2 Weeks</td>
</tr>
<tr>
<td>Sprint 3</td>
<td>User Stories and changes</td>
<td>2 Weeks</td>
</tr>
<tr>
<td>Sprint 4</td>
<td>Final changes / Features / Clean up and prepare for delivery</td>
<td>2 Weeks</td>
</tr>
</tbody>
</table>

**User Stories**

- See ERCP - Use Cases v1 Document

**Resources from Ericsson**

- Requirements / Feedback group - a team of individuals that together can provide specifics about what functionality they want to see out of the application, and give feedback throughout the development process.
- Set of typical use cases for the application.
Several developers to answer a series of questions after reviewing the source code and documentation after the project is complete.

**Technical Specifics**

The web application will be built locally on my machine using Python programming language with Django (Model View Controller framework) and traditional Html/CSS/JavaScript. The finished product will be delivered as a bundle of the source code, and MYSQL database data so that it can later be deployed on a web server.
C - Case study use case document

ERCP (Ericsson Radio Control Panel) : Use Cases

#1 - Key Stub parameter creation: Created: 20 February 2012

**Actors**: Software designer, Product owner

1. A software designer creates a new parameter in the system with as much information as he has available and marks it as a “key stub” parameter, he also selects his product owner and fills in additional information to assist the product owner in obtaining the real data. He saves the record in the system. Once the software designer has set the status to “Ready for product owner”, he is done.

2. A product owner is notified by email or logs into the system and sees that there is an outstanding key stub parameter that he has responsibility over. The product owner views the details of the key stub parameter in order to categorize and classify who he needs to contact in order to get the real values for the parameter.

3. The product owner may use the system to send an email request to the person responsible for providing the real data or he may contact them in another way. One or more outstanding parameters are selected in the system, and an email address is entered along with a message. The system generates and email with the key stub parameter details and the message entered and sends the email to the address provided.

**Tasks:**

- Add additional columns to the configParameter table and data models to support key stub parameters: boolean isKeyStub, userCreated, keyStubComments, productOwner, dateCreated, status.
- Modify system to support entering, saving and viewing of the new data into the system.
- Create user groups for software designers, and product owners.
- Create module to view outstanding key stub parameters for product owners.
- Create module to send email request with key stub details and update status.

#2 - Key Stub parameter completion: Created: 20 February 2012

**Actors**: Software designer

1. A software designer is delivered a work package that includes real data that needs to replace partial data of a key stub parameter.

2. The software designer uses the system to look up the existing key stub, and input the correct values. Then he can set the key stub to complete and save.

**Tasks:**

- Add column to the configParameter to capture the completion date of the key stub parameter.
- Modify module to save historical data and change status from key stub when the parameter is complete
- Add way to look up existing outstanding key stub parameters

#3 - Specific Parameter Search: Created: 20 February 2012

**Actors**: Software designer
1. A software designer will select a parameter key as a starting point for a specific search (will mimic what AutoGet does in the software).

2. After selecting a key, the designer will have the option to select values for a list of segment values to search against.

3. The system will conduct the search and return the 1 most correct parameter that meets the search criteria. This should match what would be returned in the software.

Tasks:

- Need to setup the data models and database schema so that it is possible to search the individual segments effectively.
- Need to create the search module where the correct information can be entered.
- Need to create the back end search logic that will find the proper parameter according to the priority of segments as will be detailed later.
D - Final software screen shots

Ericsson Radio Control Panel

Management utility for the software database

Use cases / functionality

• Parse existing software database file
• Create, view, edit, delete parameters
• Search parameters
• Key stub parameter process
• AutoGet search
• Export software database data
Key Stub process and core functionality

Software Designer Home
Creating new Parameters

Editing / updating Key stub status
Key Stub email to Product owner

ERCP - Key Stub Parameter Needs Attention

Brandon Lake,
You have been assigned a key stub parameter that needs attention:

Comments:
This is the main perf value from the new Gx platform.

Key Stub Status: Request to Product Owner
Product Owner: Brandon Lake
Created By: Brandon Lake
Created On: May 8, 2012

Click this link to receive assistance in sending a request to the responsible party:
Ercion Radio Control Panel

Thank you,
Ercion Radio Control Panel Automated Email

Product owner home
Expanded details

Product Owner request for key stub values
Request For info email

Request for radio software parameters information

To: Blake

Hi,

I am responsible for gathering data for the following key input parameter(s):

Item 1:
- Temporary Name: /nb/alpha/peak/perf
- Type: char
- Temporary Value(s): "peak Index/Cast/trace"
- Description: Index of the peak range selector.

Item 2:
- Temporary Name: /nb/alpha/peak/trim/tau
- Type: U32
- Temporary Value(s): 25455 208 32929 25539 25423
- Description:

Can you please assist me with getting the real values for these new parameters that are part of the new X2 module on the E1 platform?

Thank you,
Branden lake
slmrcake@gmail.com

Software designer completes workflow

Image of software design workflow

Image of software design workflow
AutoGet search

Autoget search results
E - Management perspective questions and responses

Describe how you felt the requirements gathering went?

Fairly well, we described the needed functionality which was implemented in accordance.

Describe how communication played a role in the success or failure of the project?

Communication is vital to formulate the requirements in specific gathering sessions as well as demos. Question if the info given in the requirements gathering sessions was sufficient or if further explanation (off line towards Albert) was required.

Describe any positive or negative actions that you observed from the way the project was completed?

Good, limited task to evaluate methodology.

After viewing the visual documentation images, describe your understanding of how the project is constructed?

Maybe some more material is needed to show the project setup, we have mainly been focused on the construction itself.

Describe how you feel the project went as a whole?

Very good, impressive how much was accomplished without further hands on support.

What would you change if you were to do this project over again?

General Comments / Feedback?

Good work!

How successful was the project in fulfilling the requirements?

How efficient was the project in completing functionality within the time allowed?
F - Technical perspective questions and responses

Describe how you felt the requirements gathering went?

I think it worked out well. We at Ericsson were a bit ill-prepared if anything. But we managed to narrow down the expectations to a number of valid user stories which were also limited in scope and hence within Brandon's control to complete. There was essentially one session where we discussed the details and agreed on the scope of the task.

Describe how communication played a role in the success or failure of the project?

Since the task was ill-defined to start with, I think that communication was central in the success of the project. Starting with the back and forth discussion to define the task, to the finer details and eventually some feedback on drafts/prototypes. We eventually ended up with a result that works for us.

After viewing the code and documentation, describe your reaction in terms of order, structure, and understanding

The code seems to be well structured into modules reflecting the use cases we decided upon. With just the user stories and next to no previous experience in Python, I can find my way through the code and inspect the critical parts. The visual documentation provided for the modules is a nice touch and makes it easier to navigate the code.

Describe how you feel the project went as a whole?

I think it was quite painless and successful. Brandon showed great interest and the final product is something we can use and continue building upon.

What would you change if you were to do this project over again?

Start with a more well defined task.

General Comments / Feedback?
How successful was the project in fulfilling the technical requirements?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

How familiar are you with the Python programming language?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

How familiar are you with the Django framework for Python?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>
How successful was the documentation in making the project structure understandable?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Bad</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>10 - Great</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

How successful was the documentation in making the code understandable?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Bad</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>10 - Great</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

Estimate how difficult or easy it would be to locate a specific portion of the code

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Difficult</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>10 - Easy</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>
Given the code, structure, and documentation, estimate how difficult or easy it would be to assume responsibility for the project as a developer.

<table>
<thead>
<tr>
<th></th>
<th>1 - Difficult</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10 - Easy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

The Project uses a ModelViewControl structure. In your opinion, how does that fact make the project easy or difficult to work with?

<table>
<thead>
<tr>
<th></th>
<th>1 - Difficult</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10 - Easy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>

In your opinion, what is the efficiency level of development using a modular style like this project?

<table>
<thead>
<tr>
<th></th>
<th>1 - Bad</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10 - Great</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

46
How satisfied are you with the final results of the project?

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Bad</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>10 - Great</td>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>