Department of Computer Science
San Marcos, TX 78666

Report Number TXSTATE-CS-TR-2011-29

REQUIREMENTS MANAGEMENT IN AN AGILE-SCRUM

Elizabeth O. Oyeyipo
Carl J. Mueller

2011-01-03

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

1 INTRODUCTION

1.1 Background on the Problem

Developing software is a difficult and extremely labor-intensive activity. As with many labor-intensive activities, developing software is error prone. Every year there are more software-based devices controlling functions that are critical to human survival. The chances of disasters and failures of these software-based devices have greatly increased. Over the past decades, several of these failures resulted in either loss of lives or property (Charlette, 2005; Leveson, 1995; Lions, 1996). Many of these development failures are attributed to software requirements engineering issues (Standish Group, 1994; Kotonya & Sommerville, 1998).

In software engineering, producing high quality software delivered within budget and schedule and satisfying stakeholders' needs is the primary objective of any software development process/project. Software requirements describe the client needs and how the software is to address them. Poor requirements and changes to requirements are one of the causes for project overrun and quality issues in the delivered software.

Many studies show that poor requirements are one of the major reasons for failed software systems and projects (Brooks, 1987; Standish Group, 1994). Worldwide, it is hard to say how many software projects failed or how much money is wasted. Defining failure as the total abandonment of a project before or shortly after it is delivered, and if one accepts a conservative failure rate of five percent, then billions of dollars are wasted each year on bad software (Charette, 2005). Some of the software failures stated below have led to significant loss of properties and lives. It is common to hear that the cause of an airliner crash or the recall of a medical device is because of undisclosed software

problems. A significant software failure is the maiden flight of the Ariane 5 rocket that ended in a crash (Lions, 1996). Another incident, of software failure that occurred between 1985 and 1987, is the Therac-25, a computer-driven medical device for delivering measured bursts of radiation to cancer patients (Leveson, 1995).

One of the most challenging aspects of the software development process is Requirements Engineering (RE). RE, the first phase of the software development process, is a critical aspect because it lays the foundation for all the subsequent project work; and it affects the success of the development project (Wiegers, 2005). The RE process consists of five main activities: Elicitation, Analysis and (Negotiation) Documentation, Validation and Management. In the software engineering field, requirements engineering has many definitions. According to Sommerville "RE is the name given to a structured set of activities that help developers to understand and document system specification for the stakeholders and engineers involved in the system development"(Sommerville, 2001). Zave defines RE as:

"Requirements engineering is the branch of software engineering concerned with the real-world goals for functions of, and constraints, on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across the software families" (Zave, 1983).

In the traditional software development methodologies, the lack of user input, incomplete requirements, and changing requirements are some of the major reasons why software systems do not deliver all their planned functionality on schedule and within budget (Weigers, 1999). In the field of software engineering, several problems became apparent in the traditional software development methodologies; and this is due to the

inflexible division of a project into separate stages. Because software developers made commitments early in the development process, it is difficult to react to changes in requirements. Having fully detailed documented requirements that will not change is unrealistic in the software development process because changes will always occur.

During the software development process, if errors occur in the requirements engineering stage and the developers continue with the project, then the customer will not be satisfied with the product. According to Boehm, it is more expensive to fix a requirements error at the later stage of the development life cycle because the cost and time required increases as the software development phase's progress (Boehm, 1981). Agile is a more recent software development methodology introduced to help address some of these system development challenges.

Agile–Scrum is an iterative development process becoming very popular in industry. However, as in all Agile methodologies, there is a resistance to the development of traditional documents. Instead of a requirements specification, Agile-Scrum employs what it refers to as a product backlog for listing stakeholder's requests. In the Agile-Scrum methodology, changing the product backlog is a normal part of the development process (Agile Alliance, 2001; Schwaber, 2001). This notion conflicts with much of the current literature on requirements engineering and management. Even in traditional development processes, requirements frequently change; but there is usually a decrease in quality and an increase in cost that is associated with the level of changes. Based on Agile methodology, growing popularity and positive reviews by developers and users, there must be some aspect of Agile-Scrum that mitigates the traditional problems associated with high levels of requirements changes.

1.2 Motivation

Agile-Scrum is an iterative framework for managing complex work, such as new product development commonly used with agile software methodologies. Change is an inherent part of Agile-Scrum. One of the most important aspects of Agile methodology is that change is a built-in aspect of the process. However, Agile-Scrum sees change in requirements as a positive aspect to the success of the software project and quality of the software product. Changes to requirements are inevitable in the software development process. There is need to manage these frequent changes so the quality of the product can be measured or to ascertain that the prioritized requirements have been implemented and traced to the source. Requirements management and Agile developments are current areas of study. Many authors have written on requirements changes; however, most recent literature has not specifically studied an empirical approach to requirements management in an Agile-Scrum development process. This research intends to demonstrate how the application of the Agile-Scrum methodology produces software that meets user needs. If the traditional notions of requirements change are correct, then using the Agile-Scrum should deliver less functionality in a fixed-length development project.

1.3 Outline of Work

The structure of this thesis is as follows: Chapter 2 provides background information on the related research. Chapter 3I contains the explanation of the research hypotheses. The description of the case study details, findings, and results are in Chapter 4. Lastly, Chapter 5 gives the conclusion and direction for future research.

2 BACKGROUND

2.1 Introduction

Many studies in requirements engineering have major areas of concentration ranging from requirements gathering to requirements specifications. Many researchers in requirements engineering are interested in validation of requirements; others have fully focused on the requirement elicitation. Study on measurement of requirements and prediction of the quality of the software product is essential. Nuseibeh and Easterbrook specified different areas of requirements engineering that had undergone research and future research work that could be further explore. One of the research areas that is seen as crucial is "requirements management - the ability, not only to write requirements but also to do so in a form that would be readable and traceable by many, in order to manage their evolution over time" (Nuseibeh & Easterbrook, 2000).

Loconsole implements an empirical study on requirements management measures and demonstrates that a subset of a set of 38 requirements measures are a good predictor of stability and volatility of requirements and change requests (Loconsole, 2004). In order to measure the quality of requirements, we must get the requirements right in these three critical areas: definition, verification, management and by applications of appropriate tools and metrics analysis techniques (Hammer, Huffman & Rosenberg, 1998). Furthermore, improvements on the development process have greatly increased. Software developers aim to produce quality software that is within budget and schedule constraints, in order to satisfy the needs of the stakeholders and users. However, there seems to be no "silver bullet" yet in having all of these without trade-offs (Brooks, 1987).

This chapter explores information on research conducted on requirements changes, with definitions of terms used in this research. The definitions include requirements changes, requirements managements, change impact analysis, and requirements traceability. In addition, it explains the software development processes-traditional and Agile methodologies.

2.2 Terminology

In this research, the following terms are used:

Requirements are defined as the specification of what the developer should implement. Requirements may be a constraint on the development process of the system. Requirements are the description of what the customer's expectation of the delivered software when the project is completed. Many authors have defined requirements in different ways. According to Institute of Electrical and Electronics Engineers (IEEE), recommendations for requirements specification are defined as:

1. A condition or capability needed by a user to solve a problem or achieve an objective;
2. A condition or capability by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents;
3. A documented representation of a condition or capability as in one (1) or two (2) (IEEE, 1998).

A *requirements change* is either a modification to an existing requirement or a new requirement that may or may not affect existing requirements. Changes to requirements often occur due to evolving needs of system stakeholders and modifications in the business environment (Kotonya & Sommerville, 2002). Managing these

requirements changes is a fundamental activity in RE (Bohner & Arnold, 1996). It also, has been a major issue in the software development process because changes are inevitable (Sommerville & Sawyer, 2000). Changes to requirements can be in the form of adding new requirements, deleting requirements and enhancing the requirements.

Requirements management is an important part of the Requirements Engineering (RE) phase. It is a continuous process, performed with the other RE activities in parallel that proceeds through all the phases of software development, and after the product is delivered (Lauesen, 2002). According to Sommerville requirements, managements can be defined as "a volatile, dynamic process which involves the understanding and controlling changes to system requirements (Sommerville 2001). Change management process is the set of activities that assess the impact and cost of changes. When changes are proposed, the impact of the changes on other requirements and the system design must be traced." Requirements management also involves tracking the status of individual requirements and tracking both backward to their origins & forward into design elements, code modules, and tests (Weiger, 2005). Managing changing requirements is a major area of focus in this research.

Goals of requirements management

The goals of requirements management are to manage changes to a set of agreed-upon requirements that have been committed to a specific product release (CMU/SEI, 1995). Requirements management help cope with the impact of changing requirements, e.g., test cases have to be adapted in order to test the implementation against the revised requirements. Developers could use this to understand the impact of requirements changes on the product quality (Heindi & Biffl, 2002).

Requirements Management Activities

Requirements Management Activities includes all activities concerned with change and version control, requirements tracing, and requirements status tracking (Paetsch, 2003). Requirement management activities entails that changes are managed during software development. Four main activities are stated below:

- controlling changes to the requirements baseline,
- controlling versions of requirements and requirements documents,
- tracking the status of the requirements in the baseline,
- managing the logical links between the individual requirements and other work products (Loconsole, 2001; Kotonya & Sommerville 2002; Lauesen, 2002).

Change impact analysis is the activity of estimating what must be modified to accomplish a change and identifying the potential consequences of that change (Arnold & Bohner, 1996). It involves identifying the impact of a requirement's change on other artifacts by tracing the requirements to define relationships between requirements source and destination.

Requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction, from its origins, through its development and specification, to subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases (Gotel & Finkelstein, 1994).

Much research in the literature on requirements changes focuses on predicting the impact in terms of cost of the changing requirements. Measuring requirements volatility-

how much the requirements are likely to change over time in the software development project is one of the good requirements management measures.

Loconsole and Börstler investigate measures of volatility in a waterfall medium-size project. The findings from the research indicate that there is high correlation between the size of requirements and total number of requirements changes (Loconsole & Börstler, 2005).

Arnold and Bohner on analyzing the impact of requirements changes suggested that early assessment is the key to addressing the impacts of changes in software projects, by predicting the effects of changes before making the changes. In addition, he specified that to measure the impact of these changes, that software stability, traceability, complexity and size are all measures that influence the impact assessment (Arnold & Bohner, 1996).

2.3 Software Development Processes

A software development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system (Pressman, 2005). Many frameworks have evolved over the years, ranging from the traditional approaches to the most recent Agile methodologies. Some of the well-known and common traditional methodologies used in the development of software projects include the following approaches: waterfall, prototyping, incremental, spiral, and Rapid Application Development. Some of the agile methodologies are SCRUM, eXtreme Programming (XP), Feature Driven Development (FDD) and others. The most popular and oldest traditional methodology used in both large and small projects is the waterfall model (Huo, Verner, Zhu, & Babar, 2004; Royce, 1987).

2.3.1 Traditional Waterfall Model

When developing software during a software development project, the following activities are performed in stepwise phases: requirements analysis, design, implementation, testing (validation), integration, and maintenance. Waterfall software development is a document-driven methodology that follows a sequential top-down approach to development of any project. Waterfall is a rigid model where all development activities are planned at the beginning of the project. This model recommends that software be develop in successive phases. Each phase of the software development process needs to be complete before the staring the next phase. At the end of each phase, an artifact in documented form is produce. At the end of the development cycle, the customer receives the entire product.

Requirements analysis phase of this approach involves initial discussions with the customer and the development team to produce a requirements definition document. The requirements definition document contains the stakeholders' requests in a natural language. It serves as a written contract between the stakeholders/customers and the development team. Software Requirement Specification (SRS) document is derived from the requirements definition document. It is a complete description of the behavior of the system to be developed. The SRSs are written in technical terms to be understood by the designers. The design phase is next in the waterfall model, and its input is the artifacts from the analysis phase. The architectural design of the system is structured into modules implementing the requirements and specifying how to build the system. The artifacts from the design stage are implements with appropriate programming language to generate source code. The development team tests the source code to validate that the

requirements have been satisfied in the software product. Since this process follows a sequential approach, changes made in any stage affect the other phases of the development process. Figure 1 below shows the waterfall development model.

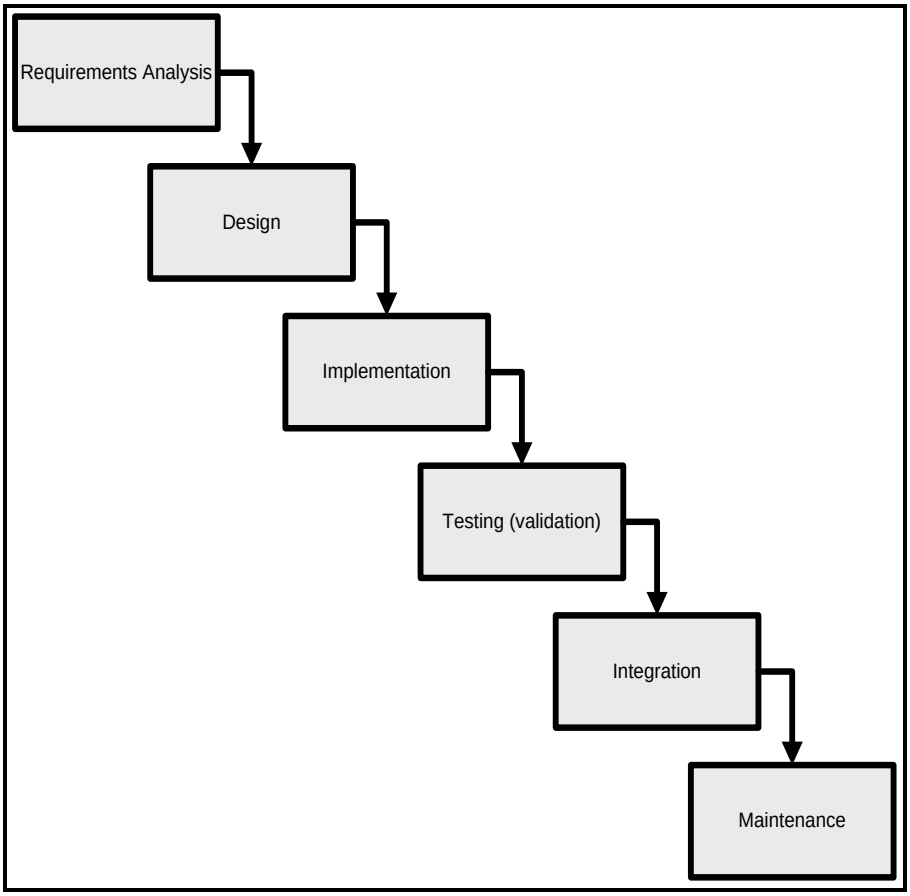


Figure 1. Waterfall Development Model (Royce, 1970).

Numerous problems are encountered using the waterfall model. According to Boehm, "the waterfall model does not adequately address concerns of developing programs and organizing software to accommodate changes. The model assumes a relatively uniform progression of elaboration steps" (Boehm, 1986).

2.3.2 Incremental Development Process

Incremental development is a refinement of the Waterfall development model. The model combines the waterfall life cycle with iterative enhancement. Incremental development involves building and validating a subset of the requirements instead of the complete requirements at once (Boehm, 1981). Figure 2 illustrates the incremental software development model. After the completion of the requirements definition, and architectural design, and implementation, the program is tested as a series of incremental builds. In each of the development increment, the model provides the customer with a subset of the product before delivery of the complete functionality. In essence, the incremental model is a scheduling technique since it does not permit developers to implement changes to the requirements.

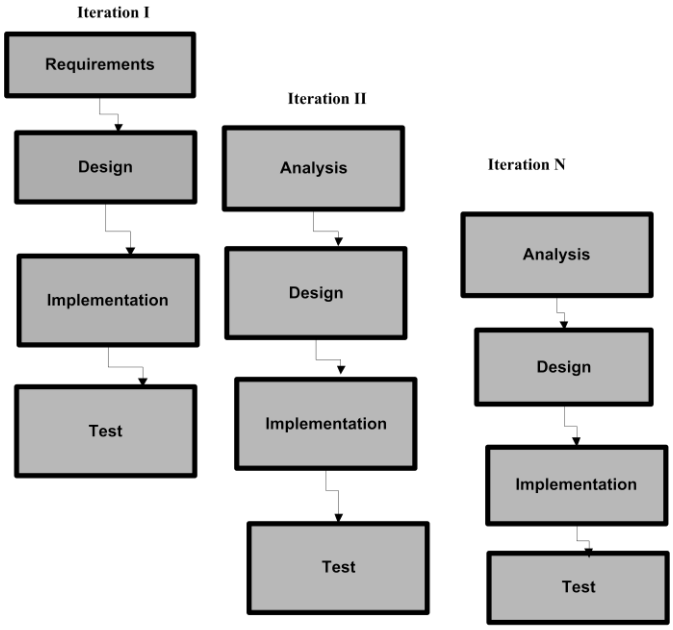


Figure 2. Incremental Development Model

2.3.3 Agile-Family of Development Methodologies

Agile software development takes a different perspective when compared to the waterfall. In February 2001, a group of consultants that developed similar methodologies

called Agile, formed the Agile Alliance and produced the Agile Manifesto (Agile Alliance, 2001; Cockburn, 2002). The following are examples of the Agile methodologies: Crystal, Dynamic System Development Method (DSDM), Extreme Programming (XP), Adaptive Software Development (ASD) and SCRUM (Cockburn, 2002). In this thesis, we explain the two most popular Agile methodologies: eXtreme Programming (XP) and SCRUM. In Agile, software development does not follow a defined process, but uses very short iterations of (2-4weeks) which focus on producing working software. Agile also allows requirements to emerge throughout the development process. Agile Manifesto based its value on the following:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

Agile software development takes a different perspective when compared to the waterfall. Agile implementation follows the key practices that support the following mechanisms:

- Iterative development – frequent iterations generate increments of work that would be inspected to determine the state of the project and serve as a basis for adaptation.
- Increment of work – composed of working system functionality rather than artifacts. These increments create a symbiotic relationship between progress and product delivery and provide a mechanism for user feedback to real product rather than arcane internal artifacts.

- Collaboration – customers and developers form teams that work together.
- Meetings – provide an internal status of the project.
- Adaptation – the teams of developers are self-organized based on the daily meetings. Developers and customers self-organize at the end of every increment to guide the project and create the greatest value.
- Emergence – the architecture, team structure, and requirements emerge during the course of the project rather than being determined at its outset. The team preliminary and sketchy vision of requirements and architecture guides the team. The architecture is initially elaborate in detail for large complex systems (Schwaber, 2002).

2.3.3.1 eXtreme Programming

eXtreme Programming is a lightweight method designed for small-to-medium sized team developing software with rapidly changing requirements (Beck, 2000). It works by bringing together the whole team in the presence of twelve simple practices, which are Planning Game, Short Releases/Frequent small releases, Metaphor, Simple design, Testing first, Refactoring, Pair programming, Collective ownership, Continuous Integration, Coding Standards, On-site Customer, and 40-hour week (Beck, 2000; Cockburn, 2002).

2.3.3.2 Agile-Scrum

Agile-Scrum follows the principles of the Agile development process. It provides the customer with the view of the product before and as each complete functionality is delivered. Agile Scrum method's main objective is to aim at prevention of common short falls in the typical traditional development process. The development teams frequently

iterate new increments of functionality. Stakeholders/product owners prioritize lists of required systems functionality, cost, timetables, and quality based on emerging business conditions. After the completed iteration, users and development teams collaborate on what to develop next, based on what was just developed and the new business needs.

Agile-Scrum is a loose set of guidelines that govern the development process of a product from its design phase to its completion. The Agile-Scrum development process recommends short iterations called sprints in two - four weeks range during which the development team makes constant trade-off decisions and adjusts to new information. During each sprint, a working, deployable version of the software is produced. Agile-Scrum recognizes that during a project the customers can change their minds about what they want due to business environment and uncertainty of what they want, by following a flexible approach to emerging requirements. In the project used for this case study, the teams follow a 14-days sprint cycle (Mueller, 2010). Figure 3 below shows the Agile-Scrum development model.

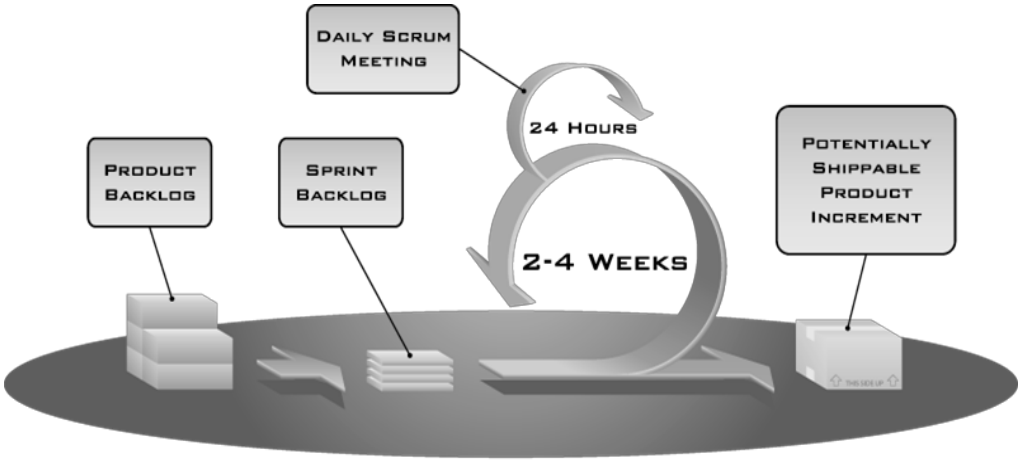


Figure 3. Agile-Scrum Development Model Mountain Goat Software (2005).

Terminologies in Agile Scrum Development:

- Product backlog- an evolving prioritized queue of business and technical functionality that needs to be develops into a system (Schwaber & Beedle, 2002).
- Product owner- an important stakeholder-demanding role may represent a larger group of products. Ensure business relevance and manages the product content.
- Scrum Master- the person responsible for the Scrum process, making sure the Scrum process is used correctly and maximizing its benefits. The scrum master acts as a coach.
- Scrum Team- is a small cross-functional group of self-organized developers responsible for actual analysis, design, implementation, and testing of the software product. Scrum team is usually not more than ten people
- Sprint- is an iteration of work during which increments of product functionality are implemented. It is normally a two to four weeks cycle.
- Sprint Backlog- contains features that the team would implement in the current sprint. These features are from the prioritized list of requirements in the product backlog. The sprint backlog holds tasks the scrum team is currently working on.
- Sprint Review- 4 hours limit period, discussing the product increment, reviewing the work that was completed and not completed.
- Sprint Planning Meeting- occurs on the first day of the sprint; the sprint backlog content is established.
- Daily Scrum Meeting- Before each sprint, the team plans a sprint. They also decide and reprioritize goals for the next sprint; they select and move features from the product backlog to the sprint backlog.

- Sprint Burn-down Chart- depicts the total task hours remaining per day. It is easy to track the status of the project progress from the burn-down chart.

2.4 Summary

This chapter examines some of the research done in requirements engineering and management, the software development methodologies, traditional, Agile methodologies, Agile-Scrum principles and terminologies used in this thesis. Earlier research on requirements changes in traditional waterfall development has found that changes affect cost, delivery schedule, and quality of the software product. The community of software engineers has both positive and negative view of evolutionary Agile-Scrum methodology that embraces requirements changes. A description of the research hypothesis is in Chapter 3.

3 RESEARCH GOALS

The primary goal of this research is to analyze the effects of requirements changes in the Agile-Scrum development process and explore the assertion that the Agile family of methodologies embraces requirements changes in a positive way. Development processes using the waterfall model as their basis recommend not implementing requirements changes until the next development cycle because of the increased development cost (Boehm, 1981). This difference leads to a number of interesting questions:

1. What makes Agile methodologies more receptive to changes even in the later phases of development?
2. Do the changes affect the development productivity?

With these questions in mind and after searching the literature on Agile methodologies, it is apparent that there is a limited amount of empirical research on the effects of requirements changes on Agile development processes.

To assist in stimulating the discussion of the effects of requirements changes in Agile methodologies, we propose to investigate these hypotheses:

I. Agile methodology allows changes to requirements even late into the project with minimal impact on software functionality and quality of the delivered product.

II. In Agile-Scrum, requirements changes do not have a significant impact on development productivity.

In the traditional waterfall development process, the cost of changes to requirements comes from having to redo the entire development cycle. To make

changes to any requirements at the later phase of the development, the developers must redo the specification and design, implement the change and test the application again. With all these changes associated artifacts like the software requirements specification, design documents are also changed.

Agile-Scrum responds to requirements changes by reducing document creation. It breaks tasks into small increments with minimal planning at the onset of the software project. Instead of planning to build the whole product from the beginning, the development teams focus on functionality that the current iteration is going to implement. For a particular sprint, iteration involves the development team working through a full software development cycle, including requirements analysis, design, coding and testing, when a working product is demonstrated to the product owner; this allows the project to adapt to changes quickly.

3.1 Agile Requirements Changes

One of the characteristics of Agile-Scrum is that it allows changes late into the project and delivers quality products on time (Agile Alliance 2001). Whereas with the Waterfall model, changes to requirements are discouraged until the current phase of development is complete (Boehm, 1981; Royce, 1970). Each of these development processes has different goals and objectives that may account for their different abilities to handle changes. Agile has its orientation in the lean manufacturing technologies pioneered by the Toyota production system (Yasuhiro, 1998; Poppendieck & Poppendieck, 2003). Its principle focus is to build only what is needed and to eliminate waste. To accomplish this objective, the Agile development methodologies strive to eliminate documentation which serves as a key factor in the waterfall process. Agile-

Scrum does create artifacts in the form of working software that serves as the criteria for the completion of each sprint cycle. In Waterfall, creation of artifacts such as SRS and design specification documents are compulsory. The SRS document serve as the criteria for completion of the requirements phase, and the design specification document serves as the completion criteria for the design phase. The software is produced at the completion of the project.

Another visible difference between Agile-Scrum and Waterfall is the development philosophy of the methodologies. A Waterfall based development methodology implements the software in the form of "horizontal slices". These "horizontal slices" are the phases of the development process (Requirements, Design, Construction, and Validation). Agile-Scrum implements the software as a series of "vertical slices" making it flexible enough to embrace requirements changes. The term vertical slice means producing a working representation of a subset of requirements during a fixed development cycle.

Figure 4 provides a contrast between how the Waterfall model and Agile-Scrum model carry out software development. As seen in the illustration, both methodologies begin by developing a set of requirements; but there the similarity ends.

In the Waterfall, the SRS document produced serves as input to the design phase. The designer translates the requirements into an architectural structure, and specifies the functionality of each architectural component. After completing the design phase, the programmer writes the code implementing the design specification. After which, the architectural components are integrated and then validated to assure that the delivered software conforms to the stated requirements.

Alternatively, Agile-Scrum combines all the development phases during the iteration, or sprint. This entails implementing the set of prioritized list of requirements or the sprint backlog. During the construction iteration or sprint, the development team designs, implements, and validates selected requirements. At the end of each construction iteration or sprint, the developers and stakeholders conduct a review to assure that the delivered software meets the selected requirements.

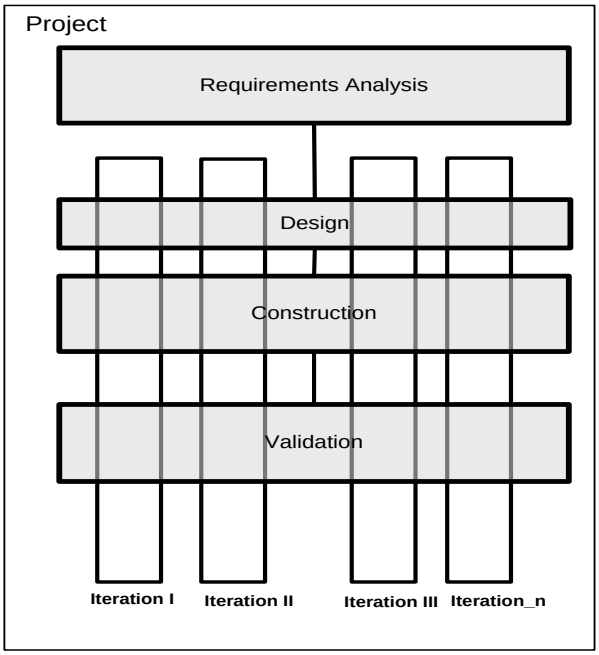


Figure 4. Agile Development vs. Waterfall

To help provide evidence for or against our hypotheses in this thesis, we design the case study using the Goal Question Metric (GQM) detailed in Appendix B to guide in collecting data from the case study. Different variables used in the metrics help to indicate the result of the stated goals. To determine if the product meets the stakeholders request in terms of the functionality delivered, the variable use is the test cases. In this research, the use of test cases is because of the relationship between the test cases and the features/requirements delivered. Because of the incremental delivery of functionality in the Agile-Scrum, requirements/features implementation may not be complete at the end of each construction iteration or sprint cycle. Using passed test cases to measure productivity provides a measure of productivity consistent with the goals and objectives

of the Agile Manifesto (Agile Alliance, 2001; Beck 2002; Harrison & Samaraweera, 1996).

Based on hypothesis I, if it is true that Agile methodology allows changes to requirements even late into the project with minimal impact on software functionality and quality of the delivered product. Then by analyzing the data collected, we expect that there would be significant differences in the level of quality and functionality of the product delivered. If the total number of test cases passed is high, then it provides confidence that a high number of functionality delivered. However, if the total number of test cases passed is low, then it is considered as a low number of functionality delivered.

One of the recommended techniques for Agile methodologies is to test before design. As the name indicates, the test before design technique requires the creation of a test case before the design process begins (Beck, 2002). Generally, when using this technique, the software test engineers create test cases to assure the implementation of each requirement during the requirements process. At the unit level of development, software test engineers create test cases before developing a function or class. Using passed test cases to monitor productivity provides an alternate to using the earned value technique. Earned value is a technique that uses the original development estimates to monitor the progress of the development. Initial program development estimates, even at the requirements estimates, are frequently very inaccurate. This is because each specific requirement in the product backlog may relate to one or more other requirements, especially with the concept of derived requirements used in this research. In addition, requirements in Agile-Scrum is not constant, developers may not implements the requirements that they start within the product backlog. Using the test cases as an

absolute measure is because before the developer delivers working software, test cases are generated for the implemented requirements.

3.2 Effects of Requirements Changes on Productivity

Another aspect of Agile-Scrum and the Waterfall approach to consider is how they both implement the requirements and changes to requirements. Agile-Scrum uses a requirement-centric approach on the stakeholder's requirements. The development teams start with a prioritized list of requirements stored in the sprint backlog. As requirements emerge and evolve due to a dynamic business environment or a stakeholder-changing request, reprioritizing of requirements in the product backlog occurs based on the current business value of the features/requirements.

Whereas, waterfall employs a document-centric approach, documentation is of tremendous importance to the success of the process. The waterfall is set with the notion that all the necessary requirements needed to complete a software product needs to be known at the beginning of the project (Sommerville, 2001). The analyst spends a much longer time in defining the customer requirements into the SRS. These requirements are considered fixed: what the team starts with at the beginning of the project is what they will implement. In Waterfall, changes to initial requirements are costly, if not impossible during the development process. Any changes made have a negative impact on the development productivity (Boehm, 1981). Changes to requirements in a Waterfall process necessitate changing the design, code, and retesting. After the completion of the project, whenever there is a need for changes, there must be a submission of a change request form to initiate any changes.

As claimed by the Agile proponents, the cost of changes to requirements do not increase the development cost. If the notion of requirements changes in traditional approaches is correct according to Boehm and Standish Group, the cost of fixing changes or errors increases exponentially as the development phases increases (Boehm, 1981; Standish Group, 1994); then in Agile-Scrum the development effort and cost should increase as the requirements changes. Based on our second hypothesis, in Agile-Scrum requirements, changes do not have a significant impact on the development productivity. The criterion used to analyze the hypothesis is, if the number of changes to requirements increases, then hours expended should increase. If the stated hypothesis II is true, we expect that the product delivered will have less functionality with high effort/cost in carrying out the product/project.

Disciplined Agile Scrum: the development process used in the project for this thesis is the disciplined Agile-Scrum. It employs the same principles of the Agile-Scrum development process with the exception that in the design phase of the development cycle, there is a detailed design history document created. Using an approach similar to development project notebook by Robert Tausworthe, the use of Microsoft notebook in the case study serves as the development teams' document history portfolio (Tausworthe, 1979). These documents serve as a reference for both current and future projects. The discipline Agile-Scrum provides the ability to track the history of code changes. It also helps to represent the project in the form of a real world presentation and helps to show the progression of the project growth.

Microsoft OneNote book: a Microsoft tool, similar to a tabbed binder, used for keeping notes, which are shared by the development teams; information is organized

section by section. Each section of the sprint notebook contains the team burn down status, backlogs, test execution, change activity and model. It also provides ease of multiple collaborations for the development team. It was used in this project to collect, organize and print reports of each section of the development process.

Development Team's Effort Estimation – Burn down Chart.

Estimating backlog effort is an iterative process that tells how long it will take for the development of the product. This estimate includes the time it takes to perform all of the requisite architecture, design construction and testing (Schwaber & Beedle, 2002). The burn down chart is a graphical chart showing the team status, and hours spent in the sprint and project. It shows the expected amount of time that the current implementation needs, as well as the expected overall progress for the entire project. A preliminary graph is created when the project is started and is updated at the end of each sprint or iteration. Generally, the left vertical axis of the burn down chart consists of the effort remaining and effort needed in terms of hours of work. The horizontal axis indicates the iteration or sprint. After completion of iteration, the amount of hours spent on each completed task for that iteration is displayed on the graph. Project managers and engineers can use this chart to gauge how much progress is being made at each step and make good estimates about how much time is needed for the completion of the project (Schwaber & Beedle, 2002 ; Cockburn, 2002; AgileAlliance).

4 CASE STUDY AND RESULTS

This section presents a detailed description of the case study used in this research. It describes the case study design, execution and the result of the data collected. In addition, outlines of the sources of possible error of the study are illustrated.

4.1 Design

A presentation of an empirical work in which the software engineering practicum class of computer science of Texas State serves as the case study for this thesis research. The study was conducted by collecting data from students developing a web-based software application in this course. There were 33 students in the class; and they formed eight teams of developers, with each team consisting of four to five members. Each team received the same problem description from the product owner to build a Resource Reservation System for scheduling resources. Disciplined Agile-Scrum is the development process used to carry out the project. A disciplined Agile-Scrum is a process that provides artifacts necessary for the development progress and regulation purposes. Illustration of the problem description of the case study is in Appendix A. To provide convincing evidence for the hypotheses stated in Chapter 3, we classify the teams into two groups:

Group 1: consists of seven teams, six employ the disciplined Agile-Scrum development process, with one team that did not make changes to the requirements until late into the project.

Group II: consists of a single control team, that employ an incremental development process; they recorded changes and modifications to the requirements but did not make changes to initial requirements.

In addition, to set a concrete measure for the data that would be collected from each group we used a popular measurement and evaluation paradigm called Goal Question Metric (GQM). A method that specifies that each organization should ensure they have some set of goals to measure, and that each goal has questions that can be quantified and answered serves as a set of metrics to ascertain if the goals are met (Basili & Rombach, 1998). Using the GQM framework on some of the checklists from Loconsole and Weiger, as well as our own list, we have a complete list of four (4) goals, eight (8) questions, and 29 measures (Loconsole, 2001; Weiger, 1999). Our four goals are as follows: to determine the total number of functionality delivered (if it meets the stakeholder's request), effort expended during the development process, to determine the impact of requirements changes, and the quality of the delivered product. The complete lists of the metrics are in Appendix B.

4.2 Case Study Execution

The following are details on the production environment used in the case study project for this research:

- Disciplined Agile-Scrum
- 14 days Sprints
- 4 sprints to the semester
- Virtual scrum Master
- Virtual product owner
- 4 - 5 person teams
- Application and Programming language used: MySQL and PHP 5.0.

The study took place during the Spring Semester, and it lasted for eight weeks, producing a total number of 32 sprints in four iterations from the development teams. The Agile–Scrum principle as used in this study takes a sprint cycle of 14 days in planning each delivery date of working software and team's portfolio. At the end of each 14-day sprint cycle, teams submitted project portfolios consisting of the following:

- Sprint Notebook: a report consisting of detailed printed information of each team's work, with the following sections: product backlogs, sprint backlogs, model, change activity report, and test execution.
- All stored product backlog in RequisitePro for the specific iteration, consisting of the proposed prioritized requirements/features request of the product owner/customer.
- All stored sprint backlog in RequisitePro for the specific iteration, consisting of the implemented prioritized requirements/features request.
- Burn Down chart: shows team status.
- Test Execution: lists the test planned, actual implemented, failed and passed, and yet to be implemented.
- Design Model : detailing the design work products stored in Rational Rose
- Source Code modules
- Working Product- in terms of deliverables

For the requirements management process of the case study project, we considered using a flexible requirements management tool that provides ease of customization and integration. A detailed description of the requirements management

tools used in this case study are specified in Appendix C. We customized the Requisite Pro with additional requirements types as specified in Figure 5 below.

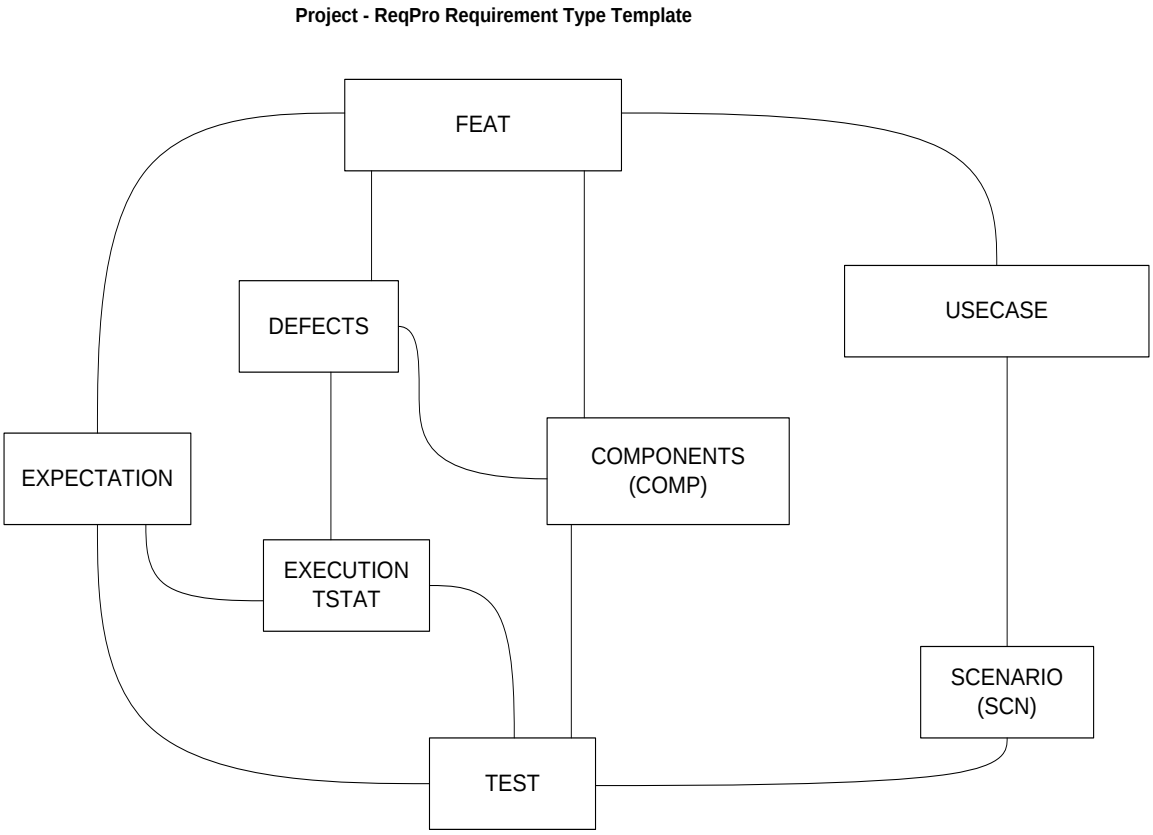


Figure 5. Customized requirements types (Mueller, 2010).

The detailed information of the specific attributes added to each requirements type are in Appendix C. These attributes and requirements types provide us with the necessary variables used in classifying the data collected from each team. These variables are requirements, defects, test cases, and effort in human hours.

We reviewed the submitted project portfolios during each sprint review. We validated the working software products by comparing them to the planned sprint backlog features in order to ascertain that the project teams are producing the right products. The teams specified their requirements from the problem description given by the product owner and stored in the RequisitePro. We asked the team to track the effort by recording

the human-effort hours expended in implementing the requirements, and when there are changes made to the requirements, recording them in the change activities section of the One Notebook. The team also reports the status and the amount of work progress in the burn down chart in the team status section of the One Notebook.

Change Introduced:

Within the third sprint, the fifth week into the project, the product owner introduced a change to the requirements. This change indicates a typical requirements change that could occur in the development environment. Based upon the data collected from the teams, we set to measure the impact of these requirements changes.

Change Request submitted by the product owner:

Expand the calendar to be able to schedule Events.

4.3 Results

We present the results of all the data collected based on the specified goals as stated above. For all Group I data, we use the average result of all data collected from the seven teams; six uses the Agile Scrum approach, while one team uses Agile with variation, by making changes to the initial requirements late into the project. Group II, consists of one team with no changes made to initial requirements. Appendix D provides detailed information for the individual teams.

Goal I: Functionality

To determine the functionality produced, we use the following two variables- test cases and sprint cycle. We counted the total number of test cases passed by finding the differences between the total number of test cases and number of failed test cases to ascertain that the functionality delivered by the teams satisfy the stakeholders' requests.

Figure 6 illustrates the functionality delivered by the two groups within the four different iterations of the scrum project. The graph illustrates Group I has the higher test cases passed; this indicates a higher functionality produced. Group II has the lower number of test cases passed. The graph indicates a slight decrease in the last two iterations of the project.

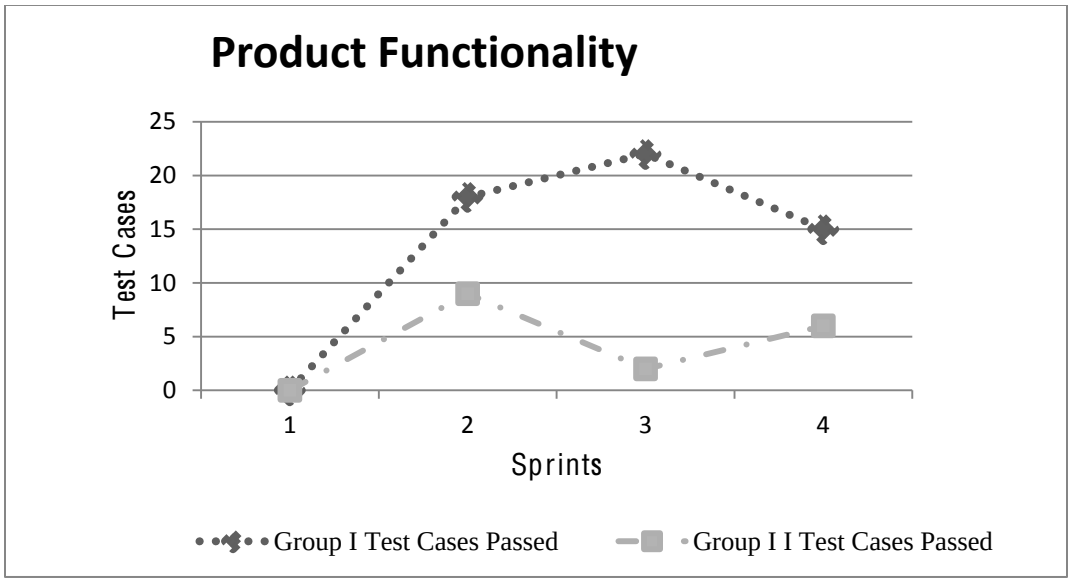


Figure 6. Product Functionality of both groups

Goal II: Development Effort

For the overall team's effort, we calculated the total number of hours reported by each team in implementing the requirements during each sprint cycle. Group I consists of the average of the seven teams. Figure 7 illustrates the hours expended in each sprint cycle. Effort expended towards the end of the project during the last sprint cycle indicates a very low difference between the two groups.

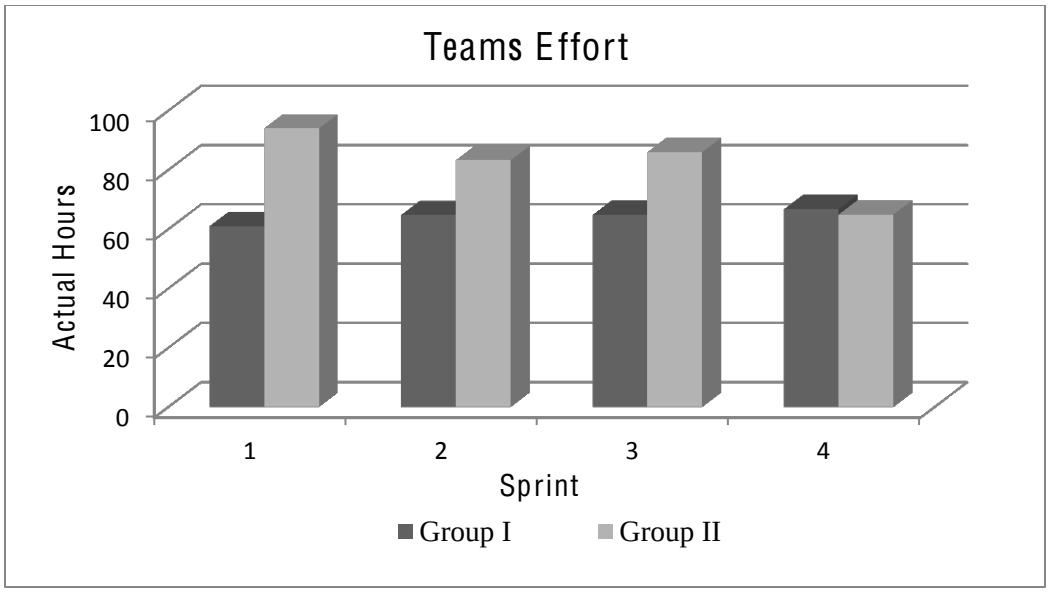


Figure 7. Development Effort for both groups

Project Progress –Based on requirements

To track the project progress, we calculated the total number of requirements and the actual total hours expended per requirements. Figure 8 illustrates the hours expended by the two groups in each sprint cycle.

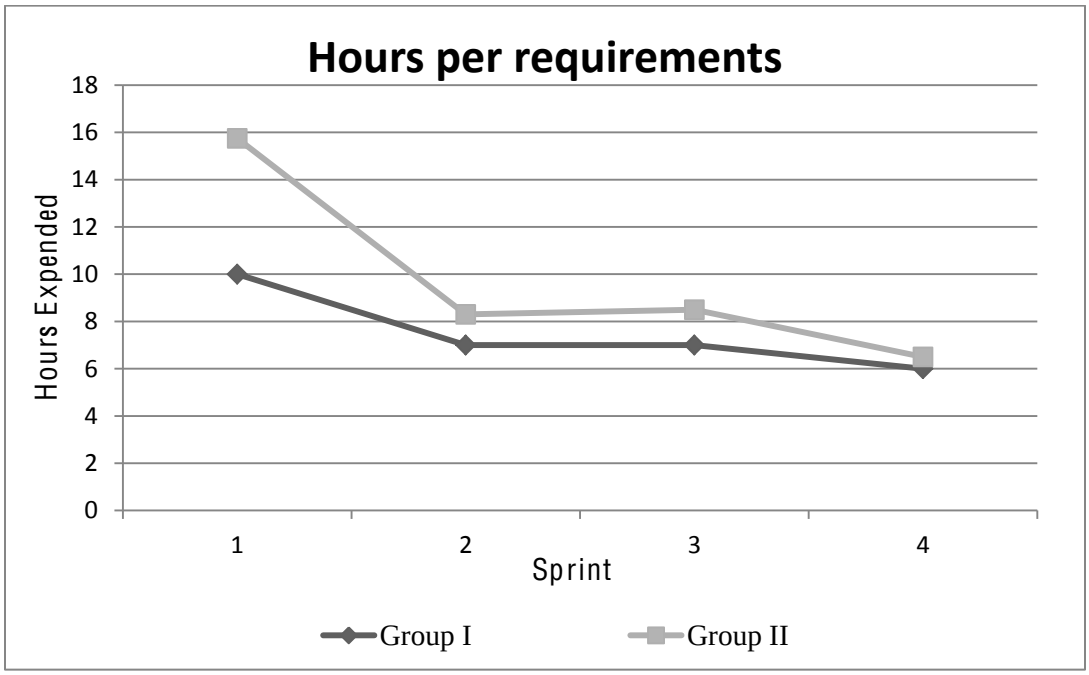


Figure 8. Project Progress for both groups

Goal III: Quality

To determine the quality of the delivered product, because it would be difficult to measure all attributes of non-functional requirements, we used defects as the variable for quality measurement. We counted the total number of defects found and the percentage of the unfixed defects in the overall project. We expected that the lower the percentage of unfixed defects the higher the level of the quality of the delivered product. Figure 9 illustrates the defects report of the two groups.

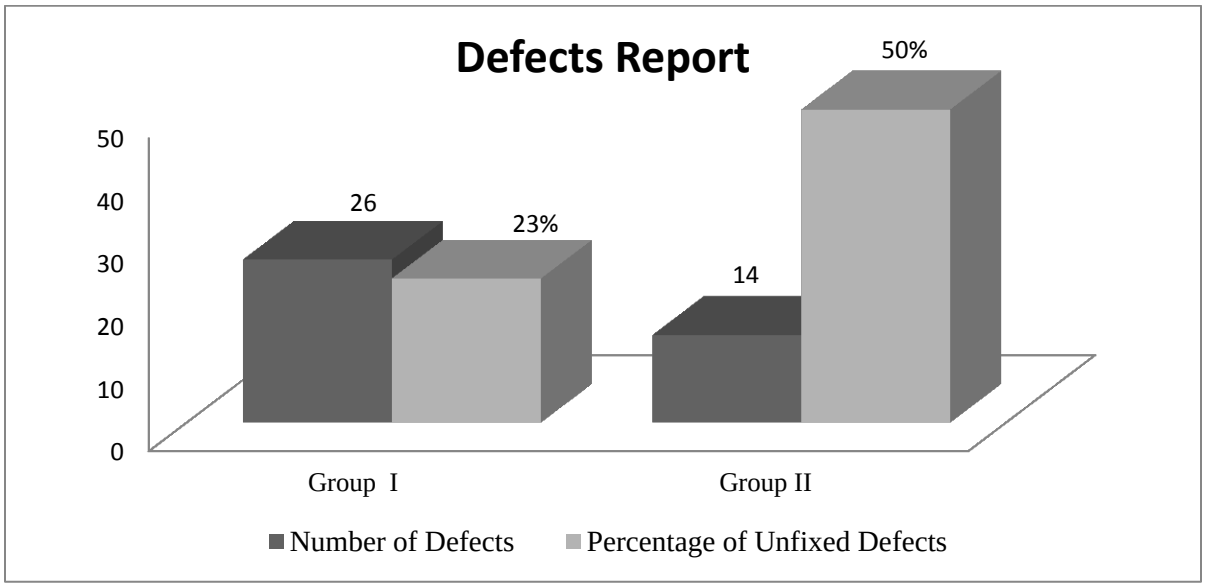


Figure 9. Defects Report for both groups.

Goal IV: Change Impact

For the requirements changes we calculated the total number of requirements changes made during the sprint cycles. All addition, deletion, and modifications to the requirements are classified as requirements changes in this study. Figure 10 illustrates the impact of requirements changes.

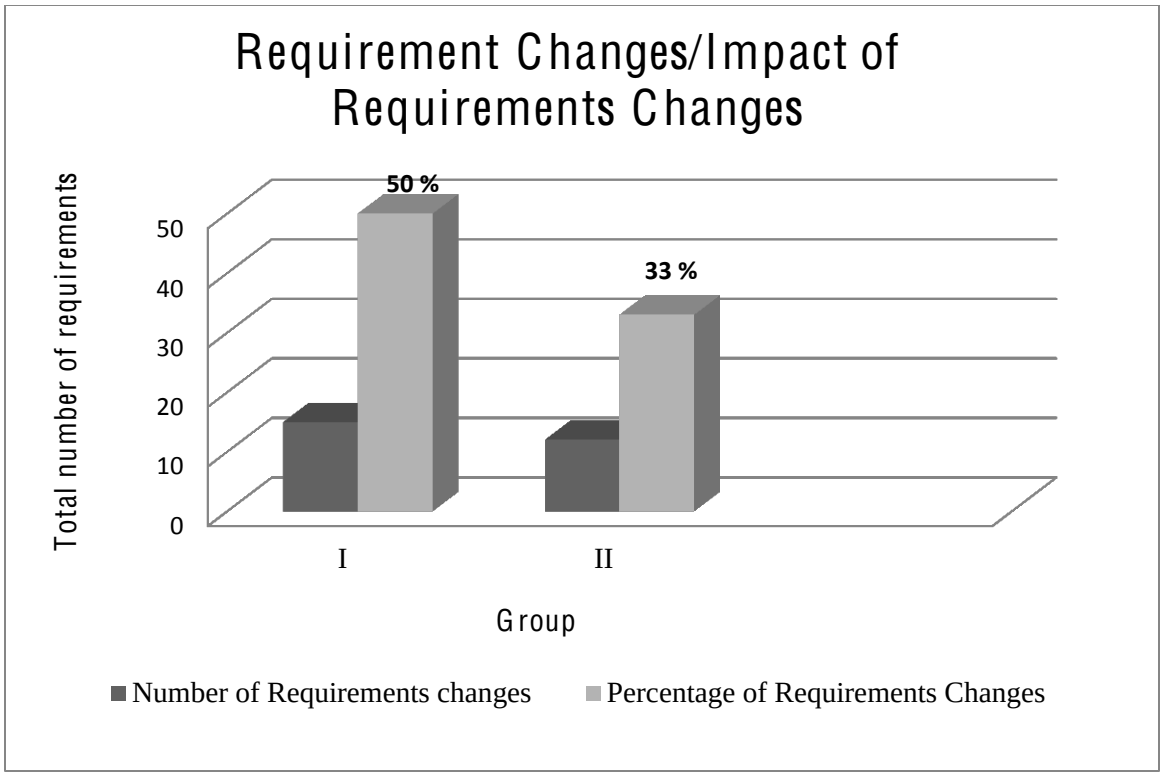


Figure 10. Requirements Change Impact.

4.4 Sources of Possible Error

Accuracy in effort reported: To get accurate reporting hours, we told the subjects that the reported hours would not affect their grades. The hours reported by the teams were used as reported, no verification of it, though we realized most of the teams recorded high hours. In addition, not all the teams reported time expended when making changes to the requirements.

Accuracy of collected data for changes: not all the teams documented the changes to requirements.

Burn down chart data: Not all the data submitted by the teams were useful, and some of the teams did not provide a breakdown of the data reported.

New Concept of Agile Development: Only a few of the students have actually done any software development work in an Agile environment. Therefore, we assume

there was a learning curve in terms of the productivity within the first sprint iteration data reported. Moreover, this would be the first time in which the practicum class would follow an Agile-Scrum software development approach for this case study.

5 CONCLUSIONS AND FUTURE RESEARCH

5.1 Conclusions

From the results of this experiment, the functionality of Group I, using Agile-Scrum, indicates more functionality delivered with higher quality, as illustrated in Figures 6 through 9. Group II makes no changes, but records the changes necessary to the initial requirements and produces less functionality. Based on these findings of this research, it appears that Agile-Scrum is more receptive to changes even in the later phases of development than an incremental process.

According to the result on the effort expended, as illustrated in Figure 7, Group II expended more hours in the first three sprint cycles with minimal differences in the fourth sprint compare to Group I. Overall Group II expended more hours on the development project, with Group I showing a uniform expenditure of effort during the development process throughout the sprint cycles.

As illustrate in Figure 8, the Agile-Scrum development groups (I) recorded more defects than the group using the incremental methods (II), but also corrected more of their defects. Suggesting that a development group using a disciplined Agile approach may produce software of higher quality. It is also not surprising that the Agile groups (I) had a higher defect count since that executed more tests that Group II.

Figure 10 illustrates that both groups develop approximately the same number of requirements, but the Agile group (I) created and implemented more changes than the Incremental group (II). Substantiating the premises that the Agile methods are more suitable to handling requirements changes.

However, from our finding, it is not clear whether the total number of hours expended has anything to do with the functionality or quality of the product delivered by the team. This is because when analyzing the individual team's result, the team with the highest number of hours expended, produced the product with the highest functionality and quality based on the results of the sprint review conducted at the end of each sprint cycle. Therefore, we conclude that the development team's performance plays a significant role in determining the functionality and quality of the delivered product.

In addition, from this experiment, using test cases to determine the functional productivity the results from our findings indicates that a high number of passed test cases correlate to high productivity and system performance as illustrated in Figure 5.

5.2 Future Research

Future research to explore would be to determine if the level/severity of changes (major, medium, low) has significant effects on the productivity. It would also be necessary to explore the effect of more requirements changes in the development process. Increasing the number of new requirements from the product owner to five different changes instead of just one used in this research.

The idea of using test cases is an appropriate measure used in literature for different purposes (Harrison & Samaraweera, 1996). In this research, a further step different from the approach used in the literature was used, by using test cases as an absolute measurement for productivity. In future research, it would be valuable to consider different measures, either by mapping the test cases to the requirements and to stakeholder's expectation.

It would also be more efficient to observe if there would be differences in the data collected from the groups, if the developers report data in a more controlled environment and log in time for each feature implemented, and for changes made. In addition, more teams would be required for comparison instead of using a single team as the controlled group. To generalize our findings, more research is required to investigate how Agile-Scrum works in other software applications.

PROBLEM DESCRIPTION FOR RESOURCE RESERVATION SYSTEM

The format of the project with the given problem statement description is given below

Situation

You have just formed a new company with three (3) associates. A marketing consultant, employed by the group, has recommended that your new company develop a Resource Reservation System (RRS).

Definition

A resource, for this project, is defined as a physical item that has an owner and usable for only one person/group at a time. Some examples of resources are: conference rooms, projectors, test equipment, laboratories, etc

Features

Some of the features for an RRS would include, but are not limited to:

1. A reservation calendar for each resource.
2. Reoccurring reservations.
3. An optional notification to a designated resource manager for each item.
4. Identification of the person requesting the resource.
5. Usage statistic by resource and person requesting the resource.
6. Option to restrict access an item.
7. Option to use an external security system.

The system must use a MySQL database and run under PHP. It will run from a Unix server

Figure 11. Project Problem Description

APPENDIX B METRIC DERIVATION

Software Metrics and tools used.

This section contains the description of metrics and software requirements management tools used in the case study and research. The paradigm called Goal Question Metric (GQM) is use to obtain data from our study. Table 1 below illustrates the complete four (4) goals, eight (8) questions, and 29 measures.

Table 1. Goals, Questions and Metrics set.

Goals	Questions	Metrics
Developers Team effort	What is the amount of effort expended for the project?	M1: total number of sprint week calendar.
		M2: total number of features/ requirements in the products backlog.
		M3: developer's hours expended during the project.
	What is the progress of the overall project?	M1: estimated burn down hours expended on the project.
		M2: actual burn down hours expended on the project.
		M3: total number of the requirements implemented in the sprint backlog.
	What is the amount of effort expended in terms of human hours in developing working source code?	M1: estimated hours expended on the current sprint cycle.
		M2: actual hours expended on the current sprint cycle.
		M3: total number implemented requirements in the sprint backlog.
Functionality produced	Are the stakeholders requirements implemented?	M1: total number of test cases in each/all sprints cycle.
		M2: total number of failed test cases.
		M3: total number of passed test cases.
	What is the status of the requirements?	M1: total number of features / requirements in the product backlog.
		M2: total number of the planned features for each sprint in the product backlog.
		M3: total number of actual features implemented in the sprint backlog.

Goals	Questions	Metrics
		M4: total number of requirements approved.
Impact of Requirements changes		M5: Number of requirements rejected.
	What types of changes are made to the requirements?	M1: total number of requirements added.
		M2: total number of modified requirements in each sprint cycle.
		M3: total number of deleted requirements in each sprint cycle.
		M4: total number of requirements changes in the sprint/project (M1, M2 and M3).
	What is the amount of effort expended in making changes to the requirements?	M1: total number of estimated hours expended to make changes to the requirements.
		M2: total number of actual hours expended to make changes to the requirements.
Quality of the delivered product	Is the product of high quality?	M1: total number of defects submitted.
		M2: total number of defects fixed.
		M3: total number of unresolved defects.
		M4: total number of test cases in each/all sprints.
		M5: total number of failed test.
		M6: total number of passed test.

APPENDIX C COMPUTER ASSISTED SOFTWARE ENGINEERING TOOLS

In the case study for the implementation of the project, different features were considered for the tools to use. A flexible requirements management tool that provides facility for easy customization to accept Agile–Scrum attributes of the requirements type used. Secondly, it is necessary to have a tool in which the development team can work remotely for easy collaboration since it will be difficult to gather the teams to work on the project at the same time and in the same place. In addition, the tool must have the ability to provide requirements traceability, and impact analysis on the stored attributes of the requirement type. Another feature considered was a tool that enables easy integration with other tools. In this research, IBM Rational RequisitePro was our choice of requirements managements tool for the project, because it enables easy integration with other IBM Rational Suites. The suites consisted of ClearQuest, ClearCase, and Rational Rose.

IBM Rational RequisitePro is a requirements management tool that helps teams to define and manage requirements. It provides and improves communication, enhances collaboration, and is easy to customize. It is use to manage and trace a project's requirements. It integrates a database and Microsoft Word for the requirements development (IBM Rational). For this research, RequisitePro was customized to reflect requirements types used in the Agile-Scrum environment so that information needed to track requirements changes and traceability details is easy to record. The proposed prioritized requirements/features are stored in the RequisitePro. Table 2. Illustrate the requirements types added to the default templates. In addition, attributes added or

modified to each requirements type in the RequisitePro for the Agile-Scrum project are specify in Table 2 in bold.

Table 2. Requirement Types and Attributes

Default Use Case Project Template	Agile Use Case Project Changes
FEAT – Features 1. Priority: High, Medium, Low. 2. Type: Functional (non Use Case), Usability, Reliability, Performance, Supportability, Design constraint, implementation Re, Physical Re, Interface. 3. Status: Proposed, Approved, Incorporated, Validated. 4. Difficulty: High, Medium, Low. 5. Stability: High, Medium, Low. 6. Risk: Schedule -High, Medium, Low: Technology - High, Medium, Low. 7. Planned Iteration: (Type: Integer) - default value. 8. Actual Iteration: (Type: Integer) - default value. 9. Origin: Helpdesk, partners, competition, Large Customers and End-users. 10. Contact Name: (Type: Text) – default value. 11. Enhancement Request: (Type: Clear Quest) - default value. 12. Defect: (Type: Clear Quest) - default value. 13. Obsolete: True, False (default).	FEAT 1. Priority: High, Medium, Low 2. Type: Functionality, Reliability, Efficiency, Usability, Maintainability, Portability 3. Status: Proposed, Approved, Incorporated, Validated 4. QFD: expected, normal, exciting 5. Risk: Schedule - (High, Medium, Low), Technology – (High, Medium, Low) 6. Estimated Time: (Type: integer) 7. Actual Time: (Type: integer) 8. Planned Sprint: (Type: Integer) - default value 9. Actual Sprint: (Type: Integer) - default value 10. Enhancement Request: (Type: Clear Quest) - default value 11. Defect: (Type: Clear Quest) - default value 12. Obsolete: True, False (default)
STRQ: Stakeholders Request 1. Stakeholder Priority: High, Medium, Low. 2. Origin: Helpdesk, partners, competition, Large Customers and End-users.	None

Default Use Case Project Template	Agile Use Case Project Changes
SUPPL: Supplementary RQ 1. Priority: High, Medium, Low. 2. Status: Proposed, Approved, Incorporated, Validated. 3. Difficulty: High, Medium, Low. 4. Stability: High, Medium, Low. 5. Risk: Schedule -High, Medium, Low: Technology - High, Medium, Low. 6. Contact Name: (Type: Text) – default value. 7. Enhancement Request: (Type: Clear Quest) - default value. 8. Defect: (Type: Clear Quest) - default value. 9. Obsolete : True, False (default).	None
UC: Use Case 1. Property: Name (default), Brief Description, Basic Flow, Alternate Flow, Special RE, Precondition, Post condition, Extension point. 2. Priority: High, Medium, Low. 3. Status: Proposed, Approved, Incorporated, Validated. 4. Difficulty: High, Medium, Low. 5. Stability: High, Medium, Low. 6. Risk: Schedule -High, Medium, Low: Technology - High, Medium, Low. 7. Planned Iteration: (Type: Integer) - default value.	UC: Use Case 1. Property: Name (default), Brief Description, Basic Flow, Alternate Flow, Special RE, Precondition, Post condition, Extension point 2. Status: Proposed, Approved, Incorporated, Validated 3. Priority: High, Medium, Low 4. QFD: expected, normal, exciting 5. Risk: Schedule - (High, Medium, Low), Technology – (High, Medium, Low) 6. Estimated Time: (Type: integer) 7. Actual Time: (Type: integer) 8. Planned Sprint: (Type: Integer) - default value

Default Use Case Project Template	Agile Use Case Project Changes
8. Actual Iteration: (Type: Integer) - default value. 1. Contact Name: (Type: Text) – default value. 2. Enhancement Request: (Type: Clear Quest) - default value. 3. Defect: (Type: Clear Quest) - default value. 4. Obsolete: True, False (default). 5. Affects Architecture (True, False).	9. Actual Sprint: (Type: Integer) - default value 10. Affects Architecture (True, False)
	TEST: Test 1. Trace from RE Type: 2. Type: Functionality, Reliability, Efficiency, Usability, Maintainability, Portability. 3. Status: Failed, Passed, Pending. 4. (Priority: High, Medium, Low)
	DEF: Defects 1. Priority: High, Medium, Low. 2. Status: found, pending , fixed. 3. Priority: High, Medium, Low. 4. Date: 5. Defects Id: (Type: integer) – default value. 6. Description: (Type: text). 7. Submitted by/Source: 8. Assigned to:
	EXP: Expectation (Succeed/Failed)
	COM: Components 1. Elements 2. Name/Type 3. Components Id:
	TSTAT: Execution 1. Test ID

Default Use Case Project Template	Agile Use Case Project Changes
	2. Date Run 3. Tester 4. Status: Pass/Fail.
	SCN: Scenario 1. Property: Name (default), Brief Description, Basic Flow, Alternate Flow, Special RE, Precondition, Post condition, Extension point. 2. Status: Proposed, Approved, Incorporated, Validated. 3. Priority: High, Medium, Low. 4. QFD: expected, normal, exciting. 5. Risk: Schedule - (High, Medium, Low), Technology – (High, Medium, Low). 6. Estimated Time: (Type: integer). 7. Actual Time: (Type: integer). 8. Planned Sprint: (Type: Integer) - default value. 9. Actual Sprint: (Type: Integer) - default value. 10. Affects Architecture (True, False).
TERM: Glossary Item	TERM: Glossary Item

Rational Rose: is a customized design tool for modeling project design and code generation. It uses the Unified Modeling Language (UML) to produce visual models of the software architectural design, database application requirements. Rose provides easy integration with other IBM Rational lifecycle development tools. It supports real-time and embedded system development. In this research, it was use by the teams for the design phase to produce the following UML based diagrams: activity diagrams, class,

component, deployment, sequence, state chart, use case, collaboration, physical storage and deployment, and physical data and tables (IBM Rational).

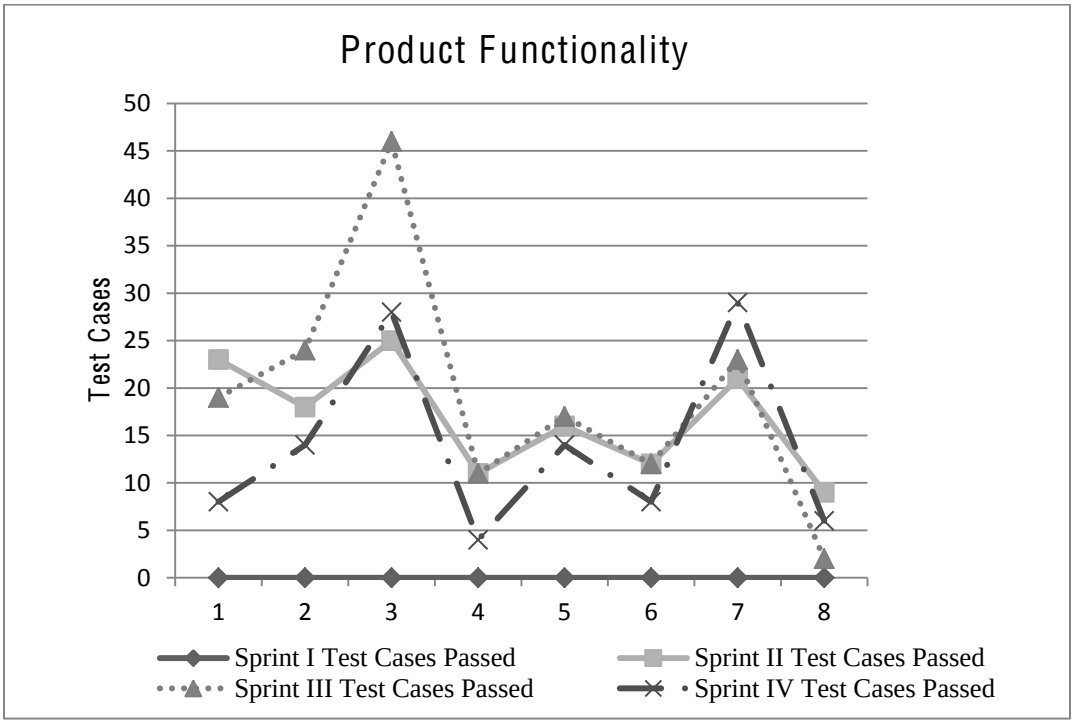
Rational Clear Quest is an automated change management tool; a defects and change tracking system designed for software development. It provides a better visibility and control of the software development lifecycle by reporting the lifecycle traceability. It provides easy integration with other IBM rational products for the requirements, development, build, test, deployment and portfolio management tools; it facilitates rapid response to changes (IBM Rational). It was use by the team for reporting the defect and enhancement made throughout each sprint cycle of the development process.

Rational Clear Case: provides sophisticated version control, workspace management, parallel development support and build auditing to improve productivity. Lightweight feature-rich clients allow the team to work remotely. It provides easy integration with other rational tools. It provides a transparent real-time access to files and directories virtually anywhere in your organization. It enables any project team size from small to large working in a distributed enterprise teams to support evolving organizational needs (IBM Rational).

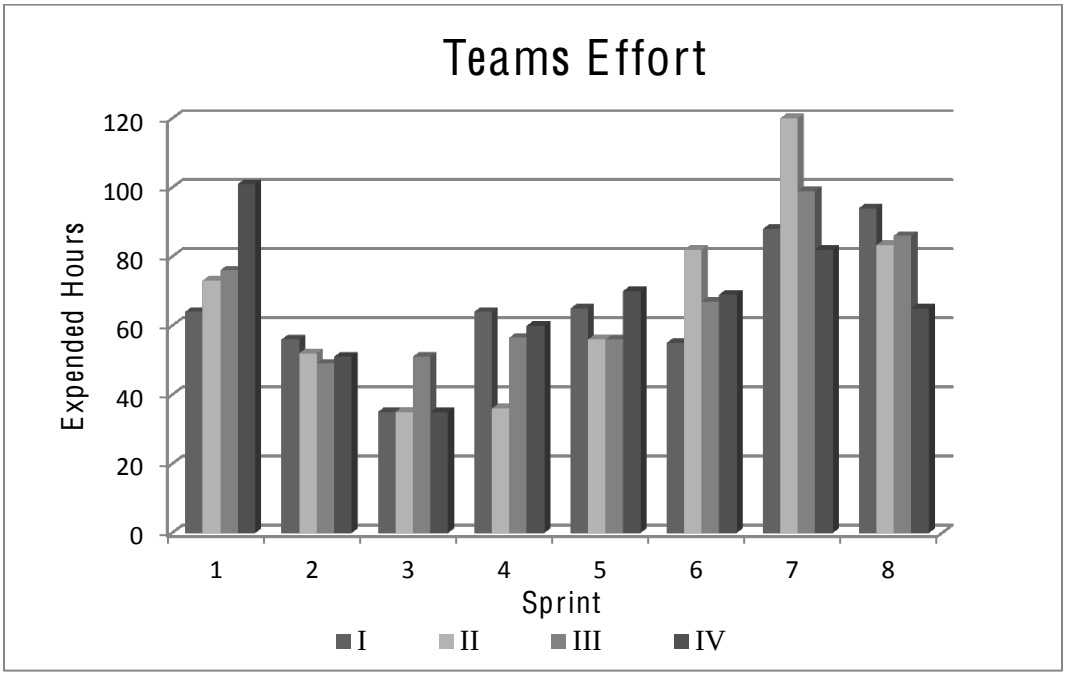
Team 2: Did not make changes until late in the project.

Team 8: Uses the incremental Approach

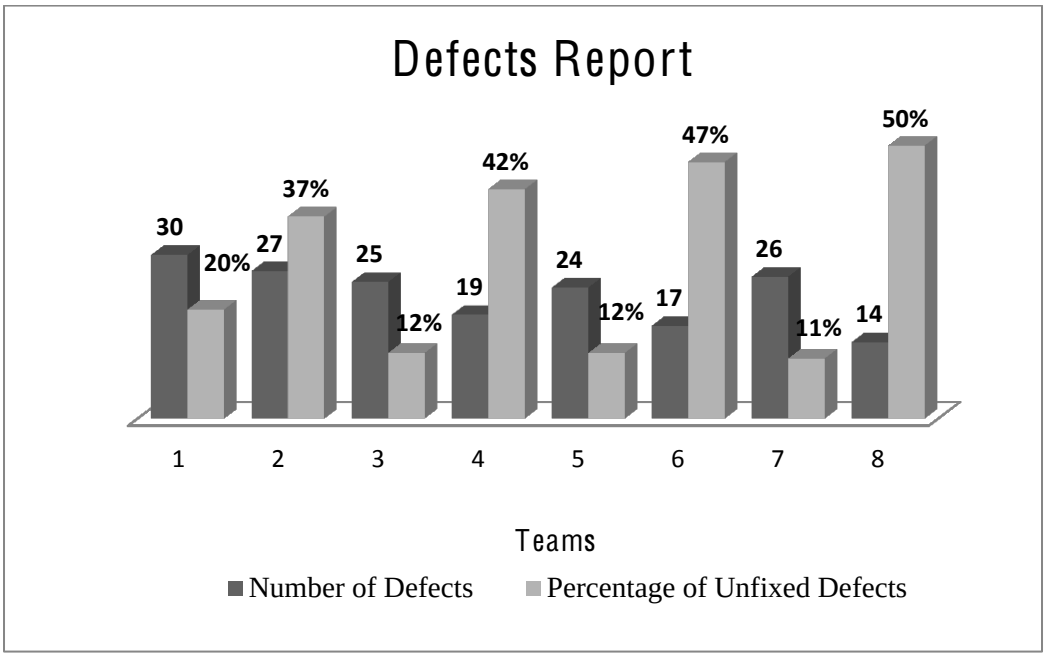
Goal I: Functionality



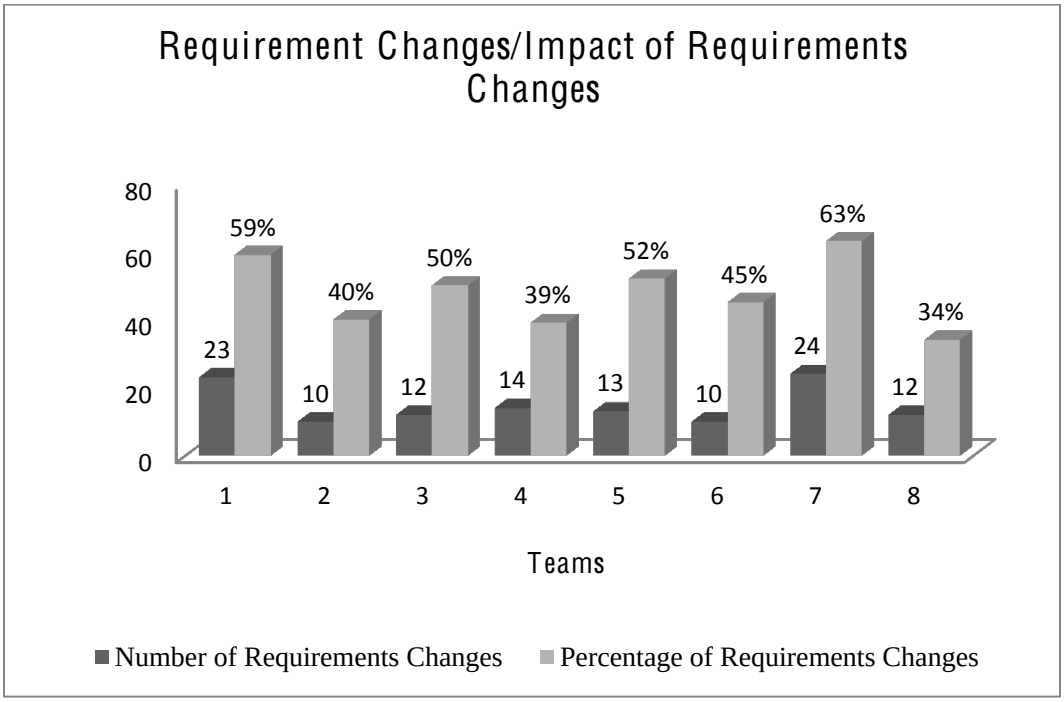
Goal II: Teams Effort



Goal III: Quality



Goal IV: Change Impact



REFERENCES

Agile Alliance. (2001) Agile Alliance URL. http://www.agilealliance.org/ Referenced July, 22nd 2010.

Arnold, R. S. & Bohner (1996). Software change impact analysis. Los Alamitos, CA, USA: IEEE Computer Society Press.

Basili, V.R. & Rombach, H.D. (1998). The TAME project: Towards. Improvement-oriented software environments in IEEE transaction on Software Engineering 14(6), pp. 758-773.

Beck, K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2000. ISBN 0201616416.

Beck, K. Test Driven Development: By Example. Addison-Wesley, 2002. ISBN 0321146530.

Boehm, B. W. (1981). Software engineering economics. Englewood Cliffs, N.J.: Prentice-Hall.

Boehm, B. (1986). A spiral model of software development and enhancement. *SIGSOFT Softw.Eng.Notes, 11*(4), 14-24. doi:http://doi.acm.org/10.1145/12944.12948.

Bohner, S. A. (2002). Software change impacts---an evolving perspective. Paper presented at the Software Maintenance, Proc. 263-272.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer, 20*, 10-19.

CMU/SEI. (1995). Carnegie Mellon University/Software Engineering Institute. The Capability Maturity Model: Guidelines for improving the Software Process. Reading, MA: Addison-Wesley.

Cockburn, A.(2002). Agile Software Development. Pearson Education, Inc.

Gotel, O. C. Z., & Finkelstein, C. W. (Eds.). (1994). An analysis of the requirements traceability problem ICRE.1994.

Hammer, P. T., Huffman, L., Linda, D., & Rosenberg, H(1998) Doing requirements right the first time! Crosstalk.

Harrison, R. Samaraweera, L.G. (1996). Using Test Case Metrics to Predict Code Quality and Effort. ACM Sigsoft pp. 78-81.

Heindl, M., (2008) Requirements tracing strategies for change impact analysis and re-testing An initial tracing activity model and industry feasibility study.

Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004). Software quality and agile methods. Paper presented at the COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference, 520-525.

IBM Rational, IBM Rational Tools http://www142.ibm.com/software/products/us/en/atoz?seltab=%23R-S, Referenced August , 20th 2010.

IEEE (1998) Std 830-1998 Institute of Electrical and Electronics Engineers,"Recommended Practice for Software Requirements Specifications." Los Alamitos, CA: IEEE Computer Society Press.

Kotonya, G., & Sommerville, I. (1998). Requirements engineering: Processes and techniques. New York: John Wiley.

Lauesen, S. (2001). Software requirements: Styles and techniques Pearson Education.

Leveson, N. (1995). Medical devices: The therac-25.

Lions, P. J. L. (1996). *ARIANE* 5, flight 501 failure, report by the inquiry board. European space agency.

Loconsole, A. (2001). Measuring the requirements management Key Process Area.

Loconsole, A. (2004). Empirical studies on requirement management measures. Software Engineering, International Conference on, , 42-44.

Loconsole, A. and Börstler J., (2005) An Industrial Case Study on Requirements Volatility Measures, in *Proceeding of APSEC — 12th IEEE Asia Pacific Software Engineering Conference*, 15–17 December 2005, Taipei, Taiwan, IEEE Computer Press, pp. 249–256.

Mountain Goat Software. (2005). http://www.mountian goatsoftware.com/topics/scrum.

Mueller, C. J. (2010). Unpublished manuscript. Texas State University–San Marcos.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. Paper presented at the ICSE '00: Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland. 35-46. doi:http://doi.acm.org/10.1145/336512.336523.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. Paper presented at the WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies.

Poppendieck M., & Poppendieck T (2003), "Lean Software Development: An Agile Toolkit", Addison-Wesley Professional, ISBN 0321150783.

Pressman, R. S. (2005). Software engineering: A practitioner's approach. Boston: McGraw-Hill.

Royce, W. W.(1970). Managing the Development of Large Software Systems, Proc. 9th. Intern. Conference.Software Engineering, ,IEEE Computer Society, 1987 ,328-338 Originally published in Proceedings. WESCON, 1970.

Schwaber, K. & Beedle, M. Agile Software Development with Scrum. Upper Saddle River, N.J Prentice -Hall, 2002.

Sommerville, I. (2001). Software engineering (6th ed.). Harlow, England ; New York: Addison-Wesley.

Standish Group (1994) "The Chaos Report", www.standishgroup.com Retrieved July, 10 2010.

van Lamsweerde, A. (2000). Requirements engineering in the year 00: A research perspective. Paper presented at the *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering,* Limerick, Ireland. 5-19. doi:http://doi.acm.org/10.1145/337180.337184.

Wiegers, K. E. (1999). Software Requirements. Redmond, WA, USA: Microsoft Press.

Wiegers, K. E. (2005). More About Software Requirements: Thorny Issues and Practical Advice. Redmond, WA, USA: Microsoft Press.

Yasuhiro Monden (1998), Toyota Production System, An Integrated Approach to Just-In-Time, Third edition, Norcross, GA: Engineering & Management Press, ISBN 0-412-83930-X.

Zave, P. (1995). Classification of research efforts in requirements engineering. Paper presented at the *RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering.*