A DMAIC Framework for Improving Software Quality in Organizations: Case Study at RK Company

Racha Karout

A Thesis

In

The Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Quality Systems Engineering) at

Concordia University

Montreal, Quebec, Canada

February, 2015

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Racha Karout

Entitled: A DMAIC Framework for Improving Software Quality in Organizations: Case Study at RK Company

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Quality Systems Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

Dr. A. Youssef	Chair
Dr. A. Awasthi	Supervisor
Dr. A. Ben Hamza	CIISE Examiner
Dr. A. Bulgak	External Examiner (MIE)

Approved by: ______________________________

Chair of Department or Graduate Program Director

___________ 2015 ______________________________

Dean of Faculty

ABSTRACT

A DMAIC Framework for Improving Software Quality in Organizations: Case Study at RK Company

Racha Karout

Managing quality is a vital aspect in software development world, especially in the current business competition for fast delivery of feature rich products with high quality. For an organization to meet its intended level of excellence in order to ensure its success, a culture of quality should be built where every individual is responsible of quality and not just the software testing team. However, delivering software products with very few bugs is a challenging constraint that is usually sacrificed in order for a company to meet other management constraints such as cost, scope and scheduling.

The purpose of this thesis is to apply six sigma DMAIC framework on 'RK' company (name anonymized) in order to help software organizations focus on improving the quality of their software products. Different phases of DMAIC methodology are applied to one of the largest software applications for 'RK' company where critical to quality aspects were identified, production bugs were classified and measured, the causes of the large number of production bugs were specified leading to different improvement suggestions. Several metrics were proposed to help 'RK' company control its software development process to ensure the success of the project under study.

ACKNOWLEGEMENT

I would like to thank my supervisor Dr. Anjali Awasthi for her guidance, help and encouragement throughout my work on this thesis.

I am also very thankful to my Manager J. D. for facilitating the interviews and collection of information. Without his help, encouragement and support, this thesis wouldn’t be completed.

I am very grateful to my family and beloved ones especially my Mom for her endless support, patience, inspiration and for always pushing me to move forward.

I would also like to use this opportunity to thank my friends and colleagues for their spiritual support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1:

INTRODUCTION

1 Background

To compete in today's world, every business needs to improve. However, improving business performance requires a structured approach, disciplined thinking, and the engagement of everyone in the organization. These elements have been the foundation for many approaches to productivity and quality improvement over the years (Evans & Lindsay, 2005).

The software industry is no exception as our world runs on software. In recent years, software has increasingly become a critical component in products in the consumer and capital goods industries. More and more products contain multiple software components (embedded systems), and depend on it for many of their functions. In addition, software has come to play an important role in many service industries such as telecoms, banking and insurance industries (Issac et al., 2010). As a result, software quality is crucial and poor quality is not acceptable. However, despite the efforts to employ software engineering methodologies, software development has not been consistently successful, often resulting in delayed, failed, abandoned, rejected software project. Even those software projects already implemented may need expensive on-going maintenance and corrective releases or service packs (Chow & Cao, 2008).

2 Problem Definition

IEEE (1991), defined software quality as the degree to which a system, component, or process meets specified requirements, in other words the degree to which a system, component, or process meets customer or user needs or expectations (as cited in Galin, 2004). The software should have few defects, since it is impossible to achieve zero defects. Chang et al. (2006) mentioned that the major software quality attributes are mainly functionality, reliability, usability and maintainability and in this case, the software should not have bugs that reduce the identified quality attributes. There should not be issues that affect its ability to maintain or re-establish its level of performance. The software should be easy to use and properly structured that would make it easy to maintain. However, with today's market competition and the need for rapid delivery, software quality is often sacrificed, thus leading to failure of the software projects. Also, the use of traditional methodology such as waterfall with the current market pace, continuously changing customer requirements, rapidly evolving technology in the current software industry plays a major role in reduced software test coverage, and as a consequence poor software quality. Moreover, people jump to solutions without fully understanding the problem or finding the root cause of poor quality.

3 Research Objectives

Based on the problems stated above, the main objectives of this research are defined as follows:

1. Improve software quality in organizations.
2. Identify the root cause(s) of the problems that result in poor quality in software organization, by identifying the boundaries of their processes, analyzing the processes that are currently followed and identifying what is critical to customer.
3. Identify improvement opportunities based on the root cause(s) of poor quality.
4. Show how companies can control their processes to ensure that the improvements applied are directing the software project towards providing the customers what they need at high level of quality.

4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 Literature Review. This chapter describes six sigma with focus on DMAIC framework. It also explains different software development processes with detailed description of agile methodology and its importance as well as other improvement tools.

Chapter 3 Solution approach. This chapter describes the proposed DMAIC framework and the benefits of DMAIC tools.

Chapter 4 Case study at RK Company. This chapter focuses on the implementation of the proposed solution on RK Company.

Chapter 5 Conclusions and Future Works. This chapter includes a SWOT analysis for six sigma DMAIC approach as well as the suggested future works.

The end of the paper contains the references and appendix containing the interview questions.

CHAPTER 2:

LITERATURE REVIEW

1 Introduction

This chapter presents the literature review on six sigma and DMAIC (define, measure, analyze, improve and control) framework. It also explains the software quality factors and identifies the common software quality problems. Different software development processes are compared and a detailed description of the agile methodology is presented with explanation of how it is used to overcome most of the identified common software quality problems. An emphasis is applied on this part since agile is considered as one of the improvement suggestions in the six sigma DMAIC framework of this paper. This is followed by a literature review of Kanban tool that is also used as an improvement suggestion and a thorough explanation of test automation is provided. Moreover, a description on how continuous improvement can improve software quality is discussed, as well as the impact of capability maturity model on certain software critical factors. Finally, a description of different management reporting metrics is presented as it has a relevance to the current paper and is used in the control phase of DMAIC methodology.

2 What is Quality?

The quest for improved quality of products, processes, and indeed, all aspects of business performance, is the driving force behind six sigma. However, people view quality in relation to differing criteria based on their individual roles in the production-marketing value chain. In addition, the meaning of quality continues to evolve as the quality profession grows and matures (Evans & Lindsay, 2005).

- Quality from the manufacturing perspective:

Garvin (1984) mentioned that manufacturing-based definition focus on the supply side of the equation, and are primarily concerned with engineering and manufacturing practice. Virtually all manufacturing based definitions identify quality as "conformance to requirements". Once a design or a specification has been established, any deviation implies a reduction in quality. Excellence is equated with meeting specifications and with "making it right the first time".

- Quality from the design perspective:

One way of defining quality is a function of specific, measurable variable and that differences in quality reflect differences in quantity of some product attribute. This assessment implies that higher levels or amount of product characteristics are equivalent to higher quality. As a result, quality is often mistakenly assumed to be related to price: the higher the price, the higher the quality, although most consumers know that this is not always true (Evans & Lindsay, 2005).

- Quality from the customer perspective:

Another definition of quality is based on the presumption that what a consumer wants determines quality. Individuals have different wants and needs and, hence, different quality standards, which leads to a user-based definition: quality is defined as fitness for intended use, or how well the product performs its intended function (Evans & Lindsay, 2005).

- Customer-Driven quality:

The most powerful customer-driven definition of quality that remains popular today: Quality is meeting or exceeding customer expectations (Evans & Lindsay, 2005).

3 Six Sigma

Six sigma can be best described as a business process improvement approach that seeks to find and eliminate causes of defects and errors, reduce cycle times and cost of operations, improve productivity, better meet customer expectations, and achieve a higher asset utilization and returns on investment in manufacturing and service processes (Evans & Lindsay, 2005). The objective of six sigma is to increase the profit margin, improve financial condition through minimizing the defects rate of product. It increases the customer satisfaction, retention and produces the best class product from the best process performance (Kabir et al., 2013). It is based on a simple problem solving methodology – DMAIC (table 1), which stands for Define, Measure, Analyze, Improve, and Control.

Sixsigma is focused on improving the basic four metrics: quality, productivity, cost and profitability (Evans & Lindsay, 2005).

D	Define
M	Measure
A	Analyze
I	Improve
C	Control

Table 1: DMAIC Framework

3.1 Six Sigma Key Concepts

The core philosophy of six sigma is based on some key concepts:

1. Think in terms of key business processes and customer requirements with a clear focus on overall strategic objectives.
2. Focus on corporate sponsors responsible for championing projects, support team activities, help to overcome resistance to change, and obtain resources.
3. Emphasize quantifiable measures that can be applied to all parts of an organization.
4. Ensure that appropriate metrics are identified early in the process and that they focus on business results, thereby providing incentives and accountability.
5. Provide extensive training followed by project team deployment to improve profitability, reduce non-value-added activities, and achieve cycle time reductions.
6. Create highly qualified process improvement experts who can apply improvement tools and lead teams.
7. Set stretch objectives for improvement.

These concepts provide a logical and disciplined approach to improving business performance, engaging the workforce, and meeting the goals and objectives of top management (Evans & Lindsay, 2005).

4 The DMAIC Methodology

The DMAIC is a process improvement cycle of six sigma program as well as an effective problem solving methodology (Hung & Sung, 2011). The five steps involved in the DMAIC methodology are described as follows:

1. Define

After a six sigma project is selected, the first step is to clearly define the problem. One must describe the problem in very specific operational terms that facilitate further analysis. A good problem statement should also identify customers and the critical to quality (CTQs) that have the most impact on product or service performance.

2. Measure

This phase of the DMAIC process focuses on how to measure the internal processes that impact CTQs. It requires understanding the causal relationship between process performance and customer value. Also, procedures for gathering facts, collecting good data, observations and careful listening must be defined and implemented.

3. Analyze

A major flaw in many problem-solving approaches is a lack of emphasis on rigorous analysis. Too often, we want to jump to a solution without fully understanding the nature of the problem and identifying the source, or "root cause", of the problem. The Analyze phase of DMAIC focuses on why defects, errors, or excessive variation occur.

4. Improve

Once the root cause of a problem is understood, the analyst or team needs to generate ideas for removing or resolving the problem and thereby improve the CTQs. This idea-gathering phase is a highly creative activity, because many solutions are not obvious.

This process includes confirming that the proposed solution will positively impact the key process variables and the CTQs, and identify the maximum acceptable ranges of these variables. Problem solutions often entail technical or organizational changes.

5. Control

The control phase focuses on how to maintain the improvements, and includes putting tools in place to ensure that the key variables remain within the maximum acceptable ranges under the modified process (Evans & Lindsay, 2005).

5 Quality Principles and Six Sigma

Modern quality management is based on three fundamental principles:

1. A focus on customers.
2. Participation and teamwork by everyone in the organization.
3. A process focus supported by continuous improvement and learning (Evans & Lindsay, 2005).

5.1 Customer Focus

The customer is the principal judge of quality. Perceptions of value and satisfaction are influenced by many factors throughout the customer's overall purchase, ownership, and service experiences. To accomplish this task, a company's efforts need to extend well beyond merely meeting specifications, reducing defects and errors, or resolving complaints. They must include both designing new products that truly delight the customer and responding rapidly to changing consumer and market demands. A company close to its customer knows what the customer wants, how the customer uses its products, and anticipates needs that the customer may not even be able to express. To meet or exceed customer expectations, organizations must fully understand all product and service attributes that contribute to customer value and lead to satisfaction and loyalty (Evans & Lindsay, 2005).

5.2 Participation and Teamwork

In any organization, the person who best understands his or her job and how to improve both the product and the process is the one performing it. When managers give employees the tools to make good decisions and the freedom and encouragement to make contributions, they virtually guarantee that better quality products and production processes will result. Employees who are allowed to participate in decisions that affect their jobs and the customer can make substantial contributions to quality and business performance (Evans & Lindsay, 2005).

The use of self-managed teams that combine teamwork and empowerment is a powerful method of employee involvement.

Six sigma relies on the participation and teamwork of employees at all levels, to understand business problems, uncover their sources, generate solutions for improvement, and implement them (Evans & Lindsay, 2005).

5.3 Process Focus and Improvement

Processes are fundamental to six sigma because, a process is how work creates value for customers (Evans & Lindsay, 2005).

Improving value added processes is the principal activity of six sigma. These improvements may take any one of several forms.

1. Enhancing value to the customer through new and improved products and services.
2. Reducing errors, defects, waste, and their related costs.
3. Increasing productivity and effectiveness in the use of all resources.

   Improving responsiveness and cycle time performance for such processes as resolving customer complaints or new product introduction.

A process focus supports continuous improvement efforts by helping to understand these synergies and to recognize the true sources of problems. Major improvements in response time may require significant simplification of work processes and often drive simultaneous improvements in quality and productivity (Evans & Lindsay, 2005).

6 Software Development Processes

Many software development methodologies have evolved overtime. Each process has its advantages and disadvantages that make it suitable for specific type of projects. Table 2 describes the most popular models.

Process	Definition	Advantages	Disadvantages
Waterfall model	It consists of several non-overlapping stages. It emphasizes planning in early stages and focuses on intensive documentation	• Easy to implement • Widely used • Identifies deliverables and milestones • Document driven • Work well on mature products and weak tams	• Idealized, does not match reality • Does not reflect iterative nature of exploratory development • Unrealistic to expect accurate requirements so early in project • Delayed discovery of serious errors • Difficult to integrate risk management • Difficult and expensive to make changes(Munassar & Govardhan, 2010)
Prototyping Model	It is the development approach of activities during software development process, the development of prototypes, i.e., incomplete versions of the software program being developed	• Gives an idea of what the final system looks like. • Enables a higher output for user • Cost effective • Assists to identify any problems with the earlier design, requirements analysis and coding activities.	• Lack of flexibility • Not suitable for large applications • Project management difficulties (Maheshwari & Jain, 2012)

Spiral Model	It is a model that focuses on risk assessment and on minimizing project risk by breaking a project into smaller segments. Each cycle involves four steps: determining objectives, evaluating alternatives, developing and verifying deliverables and planning next iteration (Maheshwari & Jain, 2012).	• High amount of risk analysis • Software is produced early in the software life cycle.	• Costly to use. • Risk analysis requires highly specific expertise • Project success is highly dependent on the risk analysis phase (Munassar & Govardhan, 2010)
Agile Development	Boehm and Turner (2003) described agile process as an iterative approach in which customer satisfaction is at highest priority as the customer has direct involvement in evaluating the software (as cited in Sharma et al., 2012)	• Adaptive to changing environment • Ensures customer satisfaction • Least documentation • Reduces risk of development	• Hard to estimate effort in large deliveries • Lack of emphasis on design documentation.

Table 2: Software Development Processes

7 Waterfall Model

Since the current methodology for the project under study is waterfall, it will be described in more details here.

Royce (1987) mentioned that the waterfall model is a sequential software development process in which progress is regarded as flowing increasingly downwards (similar to a waterfall) through a list of phases that must be executed in order to successfully build a computer software. Originally, the waterfall model was proposed by Winston W. Royce in 1970 to describe a possible software engineering practice (as cited in Bassil, 2012). The waterfall model defines several consecutive phases that must be completed one after the other and moving to the next phase only when its preceding phase is completely done. For this reason, the waterfall model is recursive in that each phase can be endlessly repeated until it is perfected (Bassil, 2012). Figure 1 describes the different phases of waterfall.

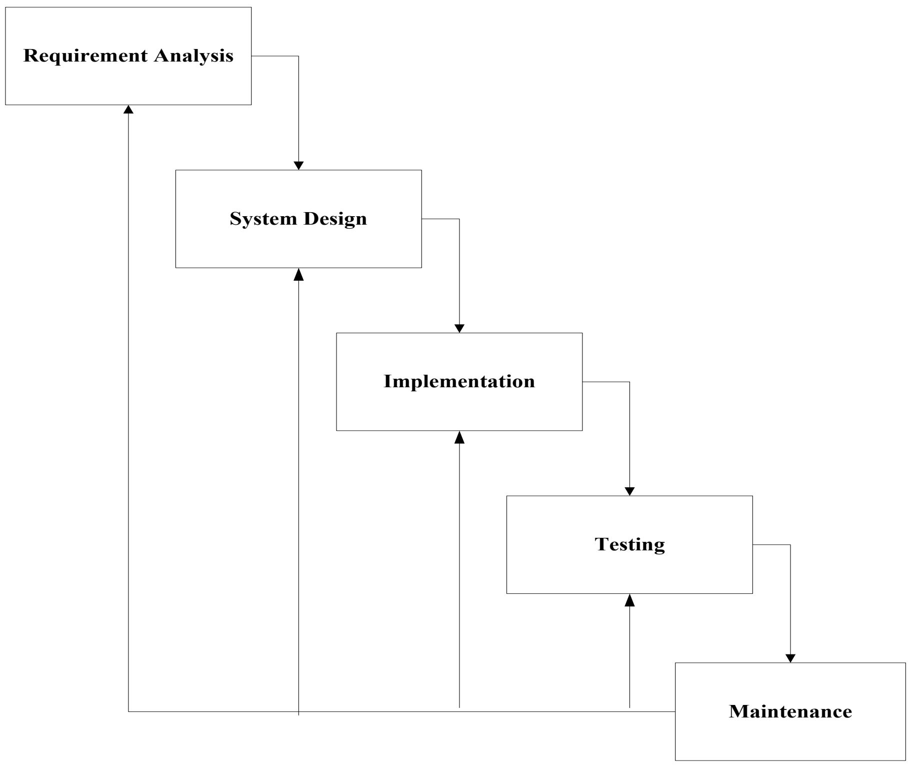

Figure 1: Waterfall model

1. **Requirement Analysis Phase:** often known as Software Requirements Specification (SRS) is a complete and comprehensive description of the behavior of the software to be developed. It implicates system and business analyst to define both functional and non-functional requirements. Usually, functional requirements are defined by means of use cases which describe the users' interactions with the software. They include such requirements as purpose, scope, perspective, functions, software attributes, user characteristics, functionalities specifications, interface requirements, and database requirements. In contrast, the non-functional requirements refer to the various criteria constraints, limitations, and requirements imposed on the design and operation of the software rather than on particular

behaviors. It includes such properties as reliability, scalability, testability, availability, maintainability, performance and quality standards.

2. **System Design Phase:** It is the process of planning and problem solving for a software solution. It implicates software developers and designers to define the plan for a solution which includes algorithm design, software architecture design, database conceptual schema and logical diagram design and graphical user interface design.

3. **Implementation:** It refers to the realization of business requirements and design specifications into a concrete executable program, database, website, or software component through programming and deployment. This phase is where the real code is written and compiled into an operational application and where the database and text files are created. In other words, it is the process of converting the whole requirements and blueprints into production environment (Bassil, 2012**).**

4. **Testing Phase:** It is also known as verification and validation which is a process for checking that a software solution meets the original requirements and specifications and that it accomplishes its intended purpose (Bassil, 2012). In IEEE-STD-610 (1991) was mentioned that verification is the process of evaluating software to determine whether the products of a given deployment phase satisfy the conditions imposed at the start of that phase, while, validation is the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements (as cited in Bassil, 2012). Moreover, the testing phase is the outlet to perform debugging in which bugs and system glitches are found, corrected, and refined accordingly.

5. **Maintenance Phase:** It is the process of modifying a software solution after delivery and deployment to refine output, correct errors, and improve performance and quality. Stellman

and Green (2005) mentioned that additional maintenance activities can be performed in this phase including adapting software to its environment, accommodating new user requirements, and increasing software reliability (as cited in Bassil, 2012).

7.1 Advantages and Disadvantages of Waterfall Model

Waterfall is easy to understand and implement. It reinforces good habits such as define-before-design, design-before-code. It identifies deliverables and milestones (Munassar & Govardhan, 2010). Also, it allows to control the project by scheduling and setting deadlines for each phase of the software development life cycle. As a result, the project is more manageable since each stage has specific deliverables at specified schedule. Also, these phases are completed one at a time.

Waterfall model is good for small projects where requirements are clearly defined and detailed at the first stage. However, waterfall model is not suitable for moderate to large projects.

The level of uncertainty and risk is very high. It is idealized and doesn't match reality well. Also, the software is delivered late in project (Munassar & Govardhan, 2010), and as a result the bugs and errors are not found until the end of the software life cycle which leads to an increased cost to fix those faults.

Moreover, waterfall is not a good model for complex and object oriented projects. Another disadvantage of waterfall model is that it is not suitable for projects where requirements change all the time which leads to a high risk.

8 Software Quality Factors

Quality can be defined as high levels of user satisfaction and low defect levels, often associated with low complexity. The quality of software is assessed by a number of variables. These variables can be divided into external and internal quality criteria. External quality is what a user experiences when running the software in its operational mode. Internal quality refers to aspects that are code-dependent, and that are not visible to the end-user. External quality is critical to the user; while internal quality is meaningful to the developer only (as cited in Hossain et al., 2013). Schulmeyer, (1998) demonstrated in the table below (table 3) a version of the software quality model (as cited in Hossain et al., 2013). This model categorized 14 quality factors in three steps of the development cycle: Quality of design, Quality of performance, Quality of adaptation. This model provides a superior structure of reference to recognize software quality (Hossain et al., 2013).

		Description
Quality of Design	Correctness	Extent to which the software conforms to its specifications and conforms to its declared objectives
	Maintainability	Ease of effort for locating and fixing a software failure within a specified time period
	Verifiability	Ease of effort to verify software features and performance based on its stated objectives
Quality of Performance	Efficiency	Extent to which the software is able to do more with less system (hardware, operating system, communications, etc.) resources
	Integrity	Extent to which the software is able to withstand intrusion by unauthorized users or software within a specified time period
	Reliability	Extent to which the software will perform (according to its stated objectives) within a specified time period
	Usability	Relative ease of learning and the operation of the software
	Testability	Ease of testing the program to verify that it performs a specified function

Quality of Adaptation	Expandability	Relative effort required to expand software capabilities and / or performance by enhancing current functions or by adding new functionality
	Flexibility	Ease of effort for changing the software's mission, functions or data to meet changing needs and requirements
	Portability	Ease of effort to transport software to another environment and / or platform
	Reusability	Ease of effort to use the software (or its components) in another software systems and applications
	Interoperability	Relative effort needed to couple the software on one platform to another software and / or another platform
	Intra-operability	Effort required for communications between components in the same software system.

Table 3: Software Quality Factors (Hossain et al., 2013)

However, Ambrose and Eynon (1998) mentioned that no attempt has been made to study the software quality and customer satisfaction from the client's point of view (as cited in Issac et al., 2010). Therefore, Issac et al. (2010) made an attempt to identify the critical factors of softwar quality from the perceptions of the clients / customers. They proposed a conceptual framework as shown in figure 2 for quality management as an instrument to measure the critical dimensions of software quality as perceived by the clients.

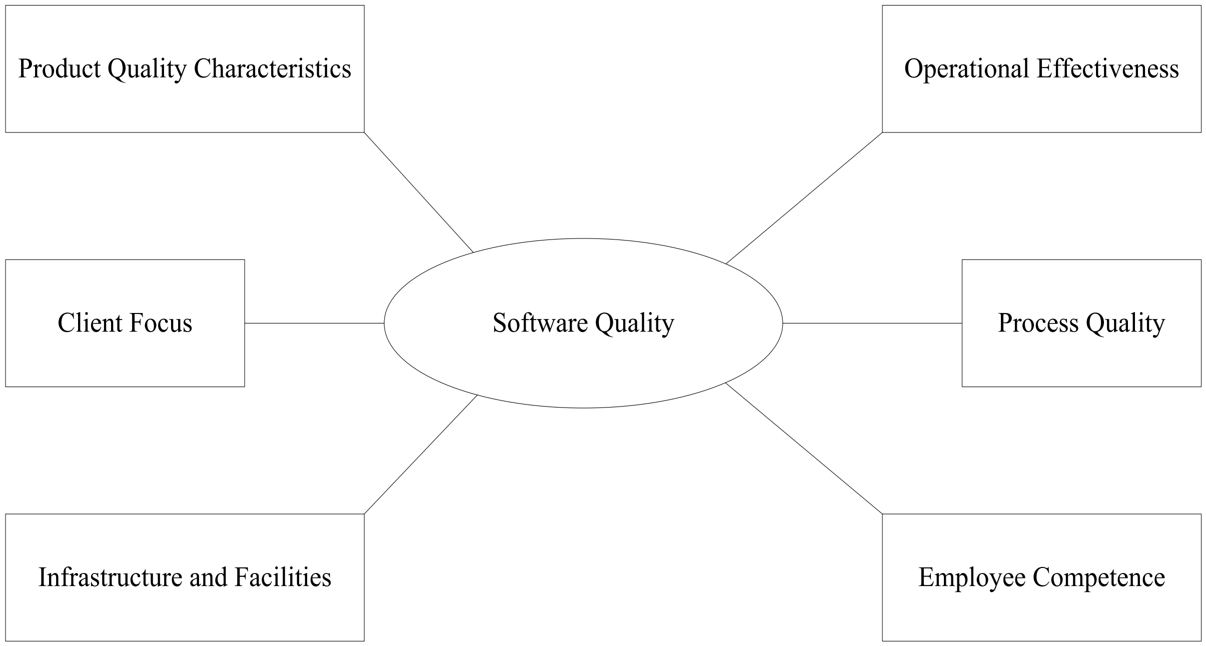
Product Quality Characteristics
Operational Effectiveness
Client Focus
Software Quality
Process Quality
Infrastructure and Facilities
Employee Competence

Figure 2: Framework of software quality from client's perspective (Issac et al., 2010)

The various characteristics classified in figure 1 are presented as follows:

Product quality characteristics

In measuring software quality, specific characteristics of a system are typically addressed. (Ben-Menachem & Marliss, 1997; Humphrey, 1989; Cho, 1998) these characteristics include flexibility, reusability, maintainability, integration, consistence reliability, functionality, efficiency and portability (as cited in Issac et al., 2010). These characteristics tend to focus on the engineering aspects of software-development which ultimately affect the user (customer or client) satisfaction (Issac et al., 2010).

Process quality management

An important issue in achieving quality is whether quality improvement and effort reduction can be simultaneously achieved. An organization's competitiveness depends on its ability to apply appropriate engineering methods and techniques to its development process, which is a key factor in software development (Issac et al., 2010). (Bunse et al., 1998; Humphrey, 1989; Li et al., 2000) Hence, to improve the product quality, the process quality needs to be improved continuously (as cited in Issac et al., 2010). Jalote (2000) mentioned that process improvement enables the same amount of software to be built in less time, with less effort and fewer defects (as cited in Issac et al., 2010).

Client Focus

The philosophy of quality management is based on customer satisfaction (Issac et al., 2010). (Ahmed, 2001; Raju & Balasubramanian, 2002) mentioned that the essence of total quality management, a management philosophy that has attracted the attention of the management fraternity in the changing global business conditions of the modern era, was to achieve customer satisfaction through continuous improvement (as cited in Issac et al., 2010). Adam et al. (2001) concluded that 'customer focus' leads to improved quality irrespective of the countries and their culture (as cited in Issac et al., 2010).

Employee competence

Boehm (1981, 1994) observed that the competence and the level of talents of personnel in the software industry were the strongest predictors of its results. The author also stated that personnel incompetence is one of the strongest project risks (as cited in Issac et al., 2010). Curtis et al. (1988) identified that the basic skill of developing software is related to managing the intellectual complexity. The authors advocated that individuals who have superior application knowledge, communication skills, high levels of motivation, team spirit and dependability are 'essential' for the success of a project (as cited in Issac et al., 2010).

Infrastructure and facilities

Jones (1998) identified that the improvement of 'support facilities' (infrastructure) was one of the essential elements of successful business performance strategies in total quality management organizations (as cited in Issac et al., 2010). Li et al. (2000) mentioned that quality (of products / service) also relies on good tools, good materials, good methods and management techniques, and latest technological developments (as cited in Issac et al., 2010). Infrastructure becomes very critical in the case of software industries, where the technological advancement is at a very rapid pace and the adaptation of technological advancement is compulsory for the survival of software organizations (Issac et al., 2010).

Operational effectiveness

The indicators of quality are related to the 'Operational effectiveness' (performance measures) of software projects (Issac et al., 2010). Harter et al., 2000 mentioned that to survive, Information Technology (IT) firms must develop high quality products 'on-time' and at low cost, i.e. 'within budget' (as cited in Issac et al., 2010). Thus it can be seen that these aspects (delivery on-time and delivery within budget) are very important measures of effectiveness and they are highly significant in achieving customer satisfaction (Issac et al., 2010).

9 Common Software Quality Problems

Some of the common software quality problems are:

1. Williams and Cockburn (2003) explained that during the project implementation, both technology and the business environment change (as cited in Stankovic et al., 2013). This leads to products that do not meet the needs of customers.
2. Customers are only involved during requirement collecting in traditional software development (Hossain et al., 2013). However, with today's market demand, customer requirements continuously change, and the lack of customer involvement and communication throughout the project, also leads to the development of projects that do not provide the required customer solutions.
3. In the traditional plan-driven software development process, work is coordinated by managers and there is a clear separation of roles (Moe et al., 2010). Thus giving the software development team less control and lose the ability of close collaboration (Stankovic et al., 2013). This will affect the performance of the team and as a result reduces the overall quality of software.
4. Software developers and testers have different mindset and goals. The developer's primary goal is to complete coding as quickly as possible, the testers' primary goal is to ensure that the software is of high quality (Yu & Petter, 2014). When these groups do not develop a shared mental model by understanding the different goals across groups, these groups will not work together to address the issues in the project and thus leads to negative impact on the project quality.
5. In the traditional software models (waterfall), the bugs and errors are not found until the end of the software life cycle, this late discovery of bugs (Monassar & Govardhan, 2010) leads to an increased cost to fix the faults. As a consequence, organizations may ignore to fix these issues as the cost is high.

The common software quality problems are summarized in table 4.

Problem	Author
Change of technology and business environment	Williams and Cockburn (2003)
Lack of customer involvement though out the project	Hussain et al. (2013)
Lack control and collaboration of software development team	Stankovic et al. (2013)
Lack of development of shared mental model between developers and testers	Yu and Petter (2014)
Late discovery of errors	Monassar and Govardhan (2010)

Table 4: Common Software Quality Problems

10 Agile Methodology

Agile methodology is one of the improvement suggestions provided in the improvement phase of the DMAIC framework for the project under study, and as a result, it will be described in more details.

Eriksson et al. (2005) define agility as follows:

Agility means to strip away as much of the heaviness, commonly associated with the traditional software-development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated deadline and the like (as cited in Dyba & Dingsoyr, 2008).

Williams and Cockburn (2003) state that agile development is about feedback and change, that agile methodologies are developed to embrace, rather than reject, higher rates of change (as cited in Byba & Dingsoyr, 2008).

In 2001, the “agile manifesto” was written by the practitioners who proposed many of the agile development methods. The manifesto states that agile development should focus on four core values:

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan (Dyba & Dingsoyr, 2008).

10.1 Important Values of Agile

Abrahamsson et al. (2002) explained the important values of agile which are:

1. First, the agile movement emphasizes the relationship and communality of software developers and the human role reflected in contracts, as opposed to institutionalized processes and development tools. In the existing agile practices, this manifests itself in close team relationships, close working environment arrangements, and other procedures boosting team spirit.
2. Second, the vital objective of the software team is to continuously turn out tested working software. New releases are produced at frequent intervals. The developers are urged to keep the code simple, straight forward and technically as advanced as possible, thus lessening the documentation burden to an appropriate level.
3. Third, the relationship and cooperation between the developers and the client is given the preference over strict contracts. From a business point of view, agile development is focused on delivering business value immediately as the project starts, thus reducing the risks of non-fulfillment regarding the contract.

4. Fourth, the development group, comprising both software developers and customer representatives, should be well-informed, competent and authorized to consider possible adjustment needs emerging during the development process life-cycle. This means that the participants are prepared to make changes and that also the existing contracts are formed with tools that support and allow these enhancements to be made.

According to Highsmith and Cockburn (2001), what is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an agile world view (as cited in Abrahamsson et al., 2002).

Miller (2001) identified agile software processes characteristics that enable shortening the life cycle of projects:

1. Modularity on development process level.
2. Iterative with short cycles enabling fast verifications and corrections.
3. Time – bound with iteration cycles from one to six weeks.
4. Parsimony in development process removes all unnecessary activities.
5. Adaptive with possible emergent new risks.
6. Incremental process approach that allows functioning application building in small steps.
7. Convergent (and incremental) approach minimizes the risks.
8. People – oriented, i.e. agile processes favor people over processes and technology.
9. Collaborative and communicative working style (as cited in Abrahamsson et al. 2002)

10.2 Critical Success Factors in Agile Software Projects

Critical success factor is defined by Bullen and Rockhart (1981) as the limited number of areas in which satisfactory results will ensure successful competitive performance for the individual, department, or organization. Critical success factors are the few key areas where “things must go right” for the business to flourish and for the managers goal to be attained (as cited in Chow & Cao, 2008).

Chow and Cao (2008) conducted a survey study that seeks to identify and provide insight into the critical success factors (CSFs) that help software development projects using agile methods to succeed. The study compiled the success factors reported in the agile literature, performed reliability analysis and factor analysis on those factors and consolidated them into a final 12 possible success factors for agile projects in five different categories: Organizational, People, Process, Technical and Project. This is shown in figure 3.

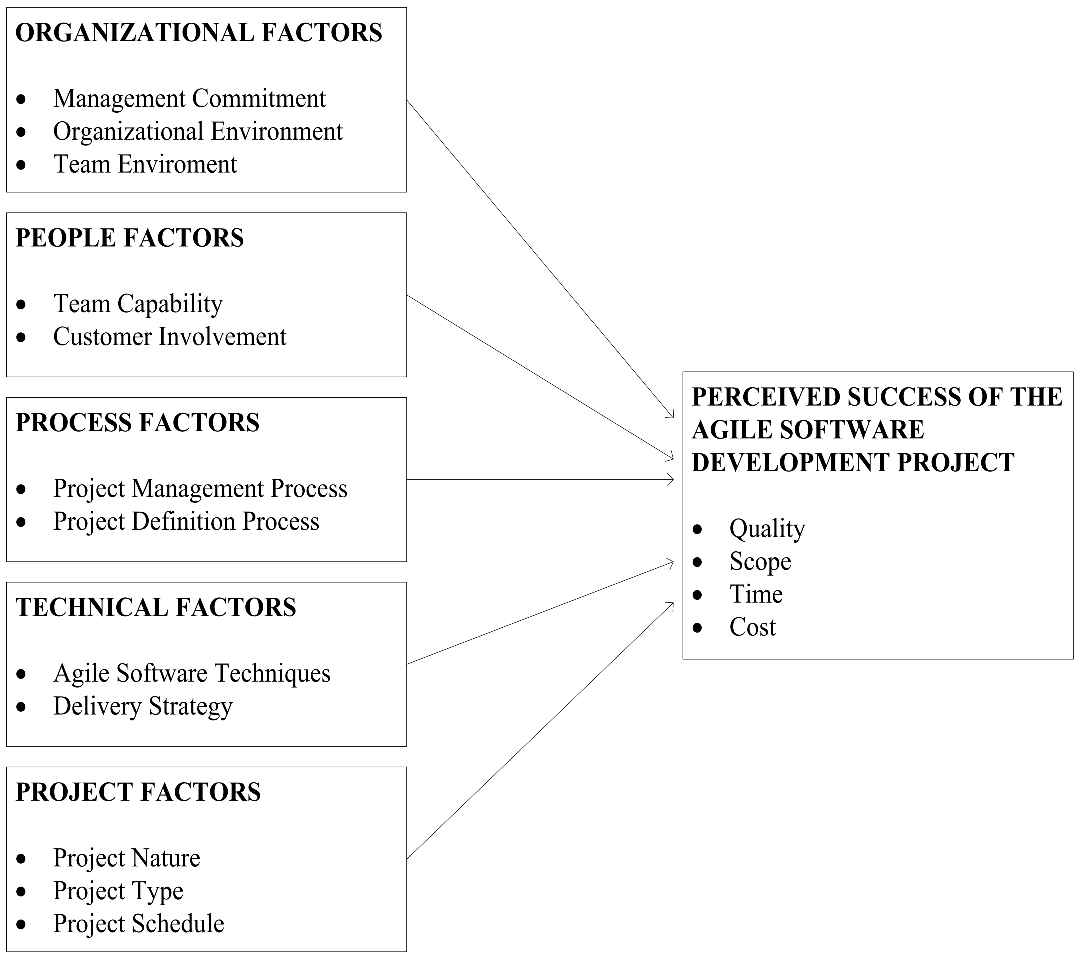

Figure 3: The research model (Chow & Cao, 2008)

In terms of attributes of success, which depict the overall perception of success of a particular project, Cohn and Ford (2003) and Lindvall et al. (2004) suggest Quality (i.e. delivering a good working product), Scope (meeting all requirements by the customer), Timeliness (delivering on time), and Cost (within estimated cost and effort) (as cited in Chow & Cao, 2008) as shown in table 5.

Dimension	Attribute
Overall perceived level of success	1. Quality (delivering good product or project outcome) 2. Scope (meeting all requirements and objectives) 3. Time (delivering on time) 4. Cost (delivering within estimated cost and effort

Table 5: Success attributes

Chow and Cao (2008) translated the 12 factors into 12 main hypothesis, each linking its existence as a critical success factor to the success of the agile software development project in terms of four success dimension: Quality, Scope, Time and Cost.

A web-based survey was conducted to gather feedback from 109 agile software projects from 25 countries around the world, and the collected data were analyzed using the multiple regression method. The analysis addressed the following questions:

a) Are these 12 factors truly the critical success factors of agile software development projects?
b) If so, what is the relative importance of each factor when compared to other factors?
c) Is there a difference among those five categories in terms of their impact on the success of an agile software development project? (Chow & Cao, 2008)

The identified critical success factors are summarized in table 6.

Critical success factors	Quality	Scope	Timeliness	Cost
1. Management commitment				
2. Organizational environment				
3. Team environment	√			
4. Team capability			√	√
5. Customer involvement		√		
6. Project management process	√			
7. Project definition process				
8. Agile software engineering techniques	√	√		
9. Delivery strategy		√	√	√
10. Project nature				
11. Project type				
12. Project schedule				

Table 6: Summary of results for first CSFs study

Chow and Cao (2008) concluded that the only factors that could be called critical success factors are found to be:

a) A correct delivery strategy

b) A proper practice of agile software engineering techniques

c) High-caliber team.

The other factors that could be critical to certain success dimensions are found to be:

a) A good agile project management process

b) An agile-friendly team environment

c) A strong customer environment

The identified success factors are summarized in table 7.

Critical success factors	1. Correct delivery strategy 2. Proper practice of agile techniques 3. High-caliber team
Critical factors to certain dimensions	1. Good agile project management process 2. Agile-friendly team environment 3. Strong customer environment

Table 7: Identified success factors

10.3 Decision-Making Challenges and Team Collaboration

Anthony (1965) described that there are three general levels of decision-making in organizations depending on the purpose of the management activity: strategic decisions, tactical decisions, and operational decisions (as cited in Moe et al., 2012). The boundaries between these levels are not always distinct. However, they differ from one another in terms of information requirements. Strategic decisions are related to organizational goals and objectives. The information concerning such decisions is usually incomplete and the decision-making process may extend over a considerable period of time. Tactical decisions are related to identification and use of resources, while operational decisions deal with ensuring effectiveness of day-to-day operations within the organization (Moe et al., 2012).

Agile software development changes the nature of collaboration, coordination and communication in software projects (Moe et al., 2012). Moe et al. (2009) mentioned that when adopting agile methods in an organization based on traditional, plan-driven development model, the focus of decision-making moves from the project manager to the software development team, and the decision-making process changes from individual and centralized to shared and decentralized. Thus, leadership is shared and important decisions on what to do and how to do it are made through an interactive process involving many people who influence each other, not just a single person (as cited in Moe et al., 2012).

Nerur et al. (2005) described that such collaborative decision-making, which involves stakeholders with diverse backgrounds and goals, is more complicated than traditional approaches, where the project manager is responsible for most of the decisions (as cited in Moe et al., 2012). Therefore, to implement agile software development successfully, it is important to explore and understand the challenges of shared decision-making.

The challenges of shared decision-making were described by designing a multiple case study consisting of four projects in two software product companies that recently adopted agile methods and more specifically scrum.

Data was collected from four projects by conducting 45 semi-structured interviews with developers, scrum masters and product owners. Also, observations of daily meetings, planning meetings, and review meetings were done. Discussions on status, progress, and how issues were perceived by team participations were done as well. All the collected information was imported into a software tool for analyzing qualitative data (NVivo).

By analyzing the data, they identified challenges of shared decision-making in agile. For example, there was often a conflict between the need for short-term progress and the need for long-term product quality at the end of sprints in three of four projects, which in turn made it difficult to align decisions on the operational level, and between the operational, tactical, and strategic levels. They also found that self-management was affected by the ability to implement a shared decision-making process. When the teams were missing a clear direction (e.g. unrealistic plans and plans without a clear priority), individual goals often become more important than team goals, and alignment among all levels seemed to fail. Introducing shared leadership and shared decision-making does not mean that everyone needs to be involved in all decisions, however, all important decisions must be communicated to the whole team, and the team needs to identify which decisions need to be taken together. Also, agile development is designed for managing project development, not for resolving company internal or cultural problems, e.g. expertise as the basis of authority (technocracy) and problems related to losing resources.

Changing the way of working is difficult, and when it involves a transition from specialized skills to redundancy of functions and rational to naturalistic decision-making, it requires a reorientation not only by the developers but also by management. This change takes time and resources, and it must be implemented to be able to succeed with agile software development. While introducing the agile approach to a software project is a top-level strategic management decision it is also important that this approach is accepted and supported by the whole organization and all stakeholders at the management and the operational levels (Moe et al., 2012).

On the other hand, Yu and Petter (2014) mentioned that agile methodology enables software development teams to adapt to customer's changing requirements through high levels of interaction and collaboration, which can lead to better project outcomes. The study focuses on answering the question: "How can theory be applied to agile software practices to explain how agile practices enable higher levels of collaboration during software development?" For this, a theory from cognitive psychology known as shared mental models was applied.

Cannon-Bowers and Salas (1993) defined shared mental model as the knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and, in turn, to coordinate their actions and adapt their behavior to demands of the task and other team members (as cited in Yu and Petter, 2014). Shared mental models provide the team with an internal knowledge base that allows team members to decide what actions to take when novel events happened (Yu & Petter, 2014). Since the purpose of this research is to explain how theory can be used to explain how agile practices create value in software development effort, they chose to examine three agile practices in depth.

1. System metaphor

Beck (1999) explained that the system metaphor is an agile software development practice in the Xtreme Programming (XP) method that is employed at the beginning of the project to develop a story that everyone –customers, programmers, and managers – can tell about how the system works. The system metaphor practice enables agile software development teams to create a "cheap" architecture design which consists of the main components of the software and their interactions (as cited in Yu & Petter, 2014). Stout et al. (1999) explained that from the lens of shared mental models theory, when an agile development team uses the system metaphor practice, they are developing a shared mental model by naturally employing the shared mental models practice of planning (as cited in Yu & Petter, 2014). The planning practice to develop shared mental models encourages teams to discuss on team goals, team roles, and how the team can react to unexpected events. The system metaphor practice is consistent with the shared mental models planning practice. The system metaphor practice encourages agile teams to create an open environment and use metaphors or stories to develop shared understandings regarding system goals, key concepts, major system functionalities, and roles and expertise of the agile team members (Yu & Petter, 2014).

2. Stand-up meeting

Paasivaara et al. (2008) explained that the stand-up meeting is one of the most basic and most frequently used scrum practices. Stand-up meetings are conducted daily and are short meetings. In this meeting, the entire team discuss the completed work, identify current bottlenecks or dependencies, and talk about next steps (as cited in Yu & Petter, 2014). Dinakar (2009) mentioned that the stand-up meeting can aid in the creation of a shared mental model within the team. Using shared mental models theory as a lens, the stand-up meeting agile practice provides opportunity to increate teams' shared understanding about task work through daily monitoring and control of the project's progress(as cited in Yu & Petter, 2014). Also, the daily stand-up meeting incorporates the shared mental models practices of leader briefings and reflexivity (Yu & Petter, 2014).

Marks et al. (2000) explained that leader briefings are a form of leader communication within teams. Leader briefings should include: (1) statement of the goals for the task, (2) identification of significant risks and how to address them, (3) specification of opportunities, and (4) prioritization of actions. Effective leader briefings conducted prior to the execution of a task enhance the similarity and accuracy on the members' mental models an increase the team's ability to adapt to changing task demands (as cited in Yu & Petter, 2014).

As for reflexivity, West (1996) defined it as the extent to which group members overtly reflect upon the group's objectives, strategies, and processes and adapt them to current or anticipated endogenous or environmental circumstances (as cited in Yu & Petter, 2014). Gurtner (2007) mentioned that using the reflexivity shared mental models practice enhances the similarity of teams' interaction models through developing a shared understanding with regard to the role of the leader in coordinating the team (as cited in Yu & Petter, 2014).

3. On-site customer

Beck (1999) explained that the on-site customer agile practice states that a customer should be present with the development team on a full time basis (as cited in Yu & Petter, 2014). Martin et al. (2010) stated that this practice requires both the developers and customers to interact daily (as cited in Yu & Petter, 2014). Koskela and Abrahamsson (2004) mentioned that the on-site customers should spend most of their time participating in planning game sessions, acceptance testing, and retrospective sessions. In addition, the on-site customer may participate in the daily stand-up meeting (as cited in Yu & Petter, 2014). The on-site customer agile practice improves the development of shared mental models, specifically, the task work mental model. The on-site customer agile practice offers developers a greater opportunity to learn the needs of the customers. The on-site customer agile practice enhances agile teams' understanding and executing stages. Through quality, frequent, and various types of communication with the customers, developers have more occasions to identify if the system's functionality meets the customers' needs and to ask questions about the system being developed. Thus, the agile team developers acquire a shared and accurate understanding of the task which enables the team to execute tasks efficiently (Yu & Petter, 2014).

10.4 How Can Agile Techniques Enhance Software Quality

Hossain et al. (2013) identified the agile techniques that enhance software quality based on the identified software quality factors that were mentioned in the software quality factors section:

1. System Metaphor

The system metaphor is a story that everyone: customers, programmers, and managers, can tell about how the system works. The idea of using a system metaphor to facilitate communication works toward revealing the reality of the team towards its task. System metaphor is helpful for communication between customer and developer. It helps the agile development team in architectural evaluation by increasing communication between team members and users. So enhance maintainability, efficiency, reliability and flexibility (Hossain et al., 2013)

2. Architectural Spike

An architectural spike is technical risk reduction techniques popularized by Extreme Programming (XP) where write just enough code to explore the use of technology or technique that you're unfamiliar with. Agile projects are designed for iteration at a time. It is a thin slice of the entire application built for the purpose of determining and testing a potential architecture (Hossain et al., 2013).

3. Onsite Customer Feedbacks

Onsite customer is one of the most practices in most agile projects that help the developers refine and correct requirements throughout the project communicating. Agile is intended to improve the software quality and responsiveness to changing customer requirements. As a type of agile software development it advocates frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted (Hossain et al., 2013).

4. Refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is also kept fully operational after each small refactoring. Practically refactoring means making code clearer and cleaner and simpler and well-designed. So refactoring reduces the probability of generating errors for the period of developments, hence improve software quality factors such as efficiency, reliability, intra-operability and interoperability, testability (Hossain et al., 2013).

5. Pair Programming

Pair programming is a technique in which two programmers or engineers work together at one workstation. One writes code while the other, the observer, reviews each line of code as it is typed in. The two programmers switch roles frequently (Hossain et al., 2013). Cockburn and Williams (2001) mentioned that while reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This procedure increases software quality without impacting time to deliver. Pair programming can improve design quality factors such as correctness, verifiability, and testability and reduce defects (as cited in Hossain et al., 2013).

6. Stand-up-Meeting

Stand-up-meeting increases the communication between team members and developers. This meeting is used to communicate problems, solutions, and promote team focus. Stand-up-meeting improve software quality factors such as reliability and flexibility (Hossain et al., 2013).

7. Continuous Integration (CI)

Fowler (2013) explained that continuous integration (CI) is a fashionable practice among agile methods where members of a team integrate their work frequently. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly (as cited in Hossain et al., 2013). This continuous application of quality control aims to improve the quality of software such as integrity, usability, testability and reduce the time taken to deliver it, by replacing the traditional practice of applying quality control after completing all development (Hossain et al., 2013).

8. Acceptance Testing

Acceptance testing refers to the functional testing of a user story by the software development team during the implementation phase. The customer specifies scenarios to test when a user story has been correctly implemented. A user story is not considered compete until it has passed its acceptance tests. This means that new acceptance tests must be created for each iteration or the development team will report zero progress. A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met. Acceptance testing occurs much earlier and more frequently in an agile methods with respect to traditional approach (Hossain et al., 2013).

Table 8 shows the quality factors that are affected by each technique.

Technique	**Affected quality factor**

System Metaphor	• Efficiency • Reliability • Flexibility • Interoperability
Architectural Spike	• Correctness
Onsite Customer Feedbacks	• Correctness • Expandability
Refactoring	• Reliability • Testability • Reusability • Interoperability
Pair Programming	• Efficiency • Testability • Portability
Stand-up-Meeting	• Flexibility
Continuous Integration	• Maintainability • Verifiability • Integrity • Usability • Expandability • Reusability • Intra-operability
Acceptance Testing	• Testability • Portability

Table 8: Agile techniques and affected quality factors

11 Agile Software Development Maturity

Fontana et al. (2014) mentioned that maturity in software development is currently defined by models such as CMMI-DEV and ISO/IEC 15504 which emphasize the need to manage, establish, measure and optimize processes. Teams that develop software using these models are guided by defined, detailed processes. However, an increasing number of teams have been implementing agile software development methods that focus on people rather than processes. As a result, they conducted a study based on the research question: “How do agile software development practitioners define maturity?” where the main objective was to identify how agile practices and the objectives of CMMI-DEV process areas are related to agile software development maturity. They collected data by forwarding the questionnaire to Brazilian agile software development practitioners. The respondents represented thirty-three different Brazilian companies and four multinational companies that developed software primarily for their own use.

In the first part of the questionnaire, the respondents had to evaluate and classify 85 agile practices on a 5-point scale ranging from 1 (No Maturity), through 2 (Somewhat Mature), 3 (Mature), and 4 (Very Mature), to 5 (Very High Maturity). In the second part of the questionnaire, in which respondents had to answer an open-ended question, they asked: “Based on your experience, what is maturity in agile software development?” They decided to use cluster analysis as a means of grouping practices according to the maturity classifications they received. They also performed a triangulation using two analysis methods: the quantitative approach for the analysis of responses of classifying agile practices and quantitative analysis of the open-ended question in the questionnaire.

<u>*Results*</u>

The highest-maturity clusters of practices and the concepts that emerged from the practitioners’ definitions enabled them to propose the following definition of agile software development maturity:

Maturity in agile software development means having an experienced team that:

- Collaborates on projects by communicating and being committed
- Cares about customers and software quality

- Allows requirements to change
- Shares knowledge
- Manages source code and tests using tools, methods and metrics supported by infrastructure appropriate for agility
- Self-organizes at a sustainable pace
- Standardizes and continuously improves agile practices
- Generates perceived outcomes for customers and management.

The quantitative analysis of the classification of the 85 practices showed that higher-maturity practices are those that support sustainable self-organization, test-driven development, caring about the solution, management of code and tests, emerging requirements and especially collaboration. These results are supported by the qualitative analysis of the answers to the open-ended question. The practitioners' concepts of maturity revealed that this is perceived mainly in the outcomes generated by the team for both management and customers. To generate those outcomes, Team and Processes play an equally important role: the process is defined and standardized by a team that collaborates and self-organizes (Fontana et al., 2014).

12 Popular Agile Methods

Some of the popular agile methods are Extreme Programming (XP) and Scrum. However, my focus will be on scrum as it is the suggested agile method for the project under study and will be explained in details.

12.1 Extreme Programming (XP)

The focus of this approach is on customer satisfaction so it empowers developers to be able to respond to changing customer requirements and to deliver high-quality software quickly and continuously. Extreme programming improves software projects by embracing communication, simplicity, feedback, respect, and courage. The original extreme programming recipe contains 12 rules: planning games, small releases, customer acceptance tests, simple design, pair programming, test-driven development, refactoring, continuous integration, collective code ownership, coding standard, metaphor and sustainability (Stankovic et al., 2013)

12.2 Scrum Method

Schwaber (1995) mentioned that the main idea of scrum is that systems development involves several environmental and technical variables (e.g. requirements, time frame, resources, and technology) that are likely to change during the process. This makes the development process unpredictable and complex requiring flexibility of the systems development process for it to be able to respond to the changes. As a result of the development process, a system is produced which is useful when delivered.

According to Schwaber (1995) scrum process has three phases: pre-game, development and post – game as described in figure 4.

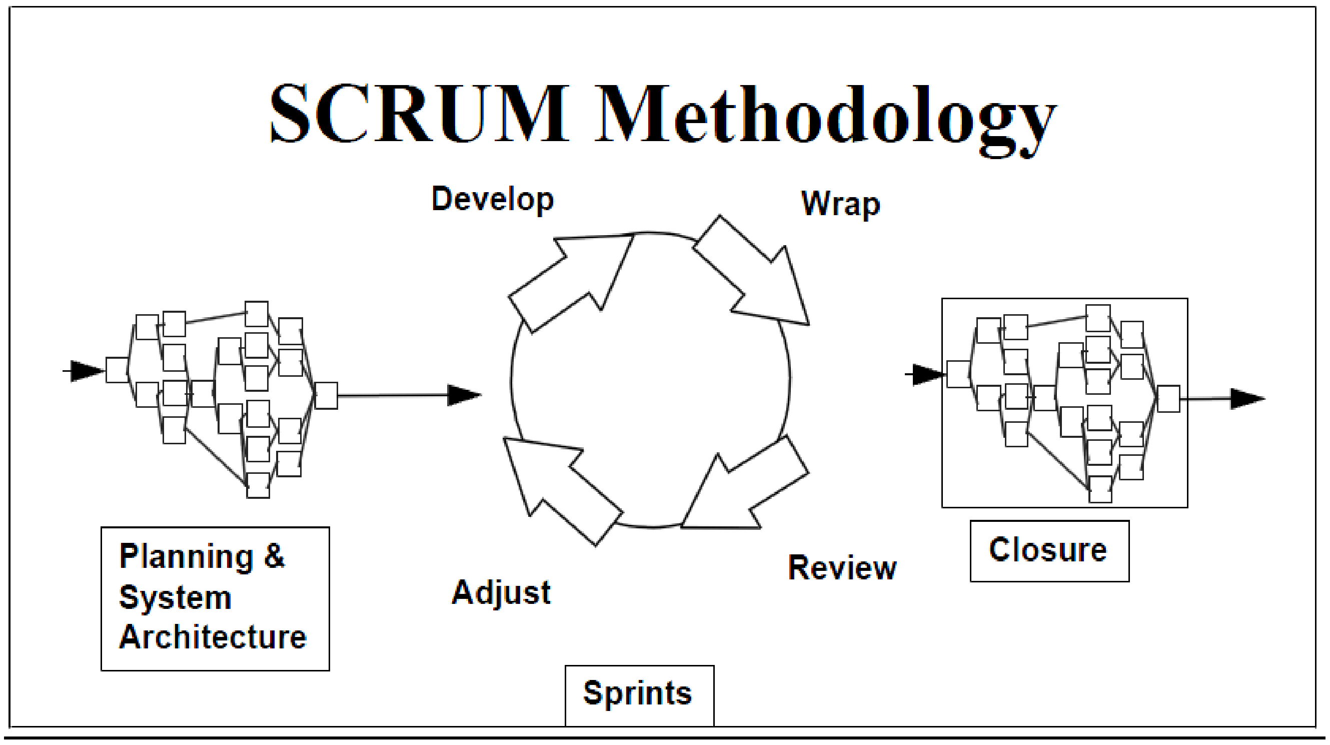

Figure 4: Scrum phases (Schwaber, 1995)

The pre-game phase: contains the planning and architecture design sub-phases. In the planning sub-phase, the list of product requirements are created in the product backlog. The items in the list are prioritized and constantly updated by adding, removing or updating the items as well as re-ordering the priorities. Information related to the resources, tools and risk assessment are also identified in the planning sub-phase. As for the architecture sub-phase, it consists of designing how the backlog items will be implements.

The Development phase: According to Schwaber (1995), this phase is an iterative cycle of development work. The management determines that time competition, quality, or functionality are met, iterations are completed and the closure phase occurs. Development consists of the following macro processes:

- Meeting with teams to review release plans.
- Distribution, review and adjustment of the standards with which the product will conform.
- Iterative sprints, until the product is deemed ready for distribution.

The sprint is an iterative cycle of development work where the scrum team organized itself to produce a new executable product increments in a sprint. Every sprint begins with the sprint planning meeting in which the product owner and the team discuss which stories will be moved from the product backlog into the sprint backlog. It is the responsibility of the product owner to determine what work the team will do and the team needs to decide how the items need to be implemented.

A sprint is a set of development activities conducted over a pre-defined period. The interval is based on product complexity, risk assessment, and degree of oversight desired.

As for the post-game or closure, according to Schwaber (1995), when the management team feels that the variables of time, competition, requirements, costs, and quality concur for a new release to occur, they declare the release "closed" and enter this phase. This phase prepares the development product for general release. Integration, system test, user documentation, training material preparations and marketing material preparation are among closure tasks.

13 What is Kanban System?

Yasuhiro (1981) explained Kanban, meaning card or marker in Japanese, is the more widely known and recognized type of pull system. A Kanban pull system is sometimes referred to as the

Toyota Production System (as cited in Marek et al., 2001). This tool is also suggested in the improvement phase of DMAIC for the project under study as it can be mixed with agile methodology.

Anderson (2010) described Kanban system as a number of cards equivalent to the (agreed) capacity of a system are placed in circulation. One card attaches to one piece of work. Each card acts as a signaling mechanism. A new piece of work can be started only when a card is suitable. This free card is attached to a piece of work and follows it as it follows through the system. When there are no more free cards, no additional work can be started. Any new work must wait in a queue until a card becomes available. When some work is completed, its card is detached and recycled. With a card now free, a new piece of work in the queuing can be started.

This mechanism is known as a pull system because new work is pulled into the system when there is capacity to handle it, rather than being pushed into the system based on demand. A pull system cannot be overloaded if the capacity, as determined by the number of signal card in circulation has been set appropriately.

Kanban quickly flushes out issues that impair performance, and it challenges a team to focus on resolving those issues in order to maintain a steady flow of work. By providing visibility onto quality and process problem, it makes obvious the impact of defects, bottlenecks, variability and economic costs on flow and throughput. The simple act of limiting work-in-progress with Kanban encourages higher quality and greater performance. The combination of improved flow and better quality helps to shorten lead times and improve predictability and due-date performance. By establishing a regular release cadence and delivering against it consistently, Kanban helps to build trust with customers and trust along the value stream with other departments, suppliers and dependent downstream partners.

The core properties that Kanban uses are:

1. Visualize workflow
2. Limit work-in-progress.
3. Measure and manage flow.
4. Make process policies explicit.
5. Use models to recognize improvement opportunities (Anderson, 2010).

13.1 Benefits of Kanban

Kanban has several benefits. It encourages the focus on quality as it has a big impact on the productivity and throughput of teams with high defect rates.

Anderson in his book (2010) suggests that collaborative analysis and design helps improve quality. When teams are asked to work together to analyze problems and design solutions, the quality is higher. He also suggests the use of design patterns to improve quality. Design patterns capture known solutions to known problems. Design patterns ensure that more information is available earlier in the lifecycle and that design defects are eliminated.

Another benefit of Kanban usage is by reducing the work-in-progress, a team can deliver more often. Anderson (2010) mentioned that reducing work-in-progress (WIP) shortens lead time. Shorter lead times mean that it is possible to release a working code more often.

Delivering small, high-quality releases builds more trust with partner teams than putting out large release less often. Small releases show that the software development team can deliver and is committed to providing value. They build trust with the marketing team or business sponsors. High quality in the released code builds trust with downstream partners such as operations, technical support, and field engineering and sales.

Another benefit of Kanban is balancing demand against throughput implies that the team can set the rate at which they accept new requirements into their software development pipe to correspond with the rate at which they can deliver working code. When they do this, they are effectively fixing their work-in-progress to a given size. As work is delivered, they will pull new work (or requirements) from the people creating demand. So any discussion about prioritization and commitment to new work can happen only in the context of delivering some existing work.

In the software development world, the Software Engineering Institute (SEI) of Carnegie Mellon University has defined the highest level of their capability Maturity Model Integration (CMMI) as optimizing. Optimizing implies that the quality and performance of the organization is continuously being refined. A work place culture where the entire work force is focused on continually improving quality, productivity and customer satisfaction is known as "Kaizen Culture".

In Kaizen culture, the work force is empowered. Individuals feel free to take action, free to do the right thing. They spontaneously swarm on problems, discuss options, and implement fixes and improvements. In a Kaizen culture, the workforce is without fear. The underlying norm is for management to be tolerant of failure if the experimentation and innovation was in the name of process and performance improvement. In a Kaizen culture, individuals are free (within some limits) to self-organize around the work they do and how they do it. Visual controls and signals are evident, and work tasks are generally volunteered for rather than assigned by a superior.

A Kaizen culture involves a high level of collaboration and a collegial atmosphere where everyone looks out for the performance of the team and the business above themselves. A Kaizen culture focuses on systems-level thinking while making local improvements that enhance overall performance. A Kaizen culture has a high level of social capital. It is a highly trusting culture where individuals, regardless of their position in the decision-making hierarchy of the business, respect each other and each person's contribution. High-trust cultures tend to have flatter structures than lower-trust cultures. It is the degree of empowerment that enables a flatter structure to work effectively. Hence, achieving a Kaizen culture may enable elimination of wasteful layers of management and reduce coordination costs as a result.

Kanban provides transparency into the work, but also into the process. It provides visibility into how the work is passed from one group to another. Kanban enables every stakeholder to see the effects of his or her actions or inactions. If an item is blocked and someone is capable of unblocking it, Kanban shows it. In addition to the visibility into process flow, work-in-progress limits also forces challenging interactions to happen sooner and more often. It isn't easy to ignore a blocked item and simply work on something else. This "stop the line" aspect of Kanban seems to encourage swarming behavior across the value stream. When people from different functional areas and with different job title swarm on a problem and collaborate to find a solution, thus maintaining the flow of work and improving system level performance, the level of social capital and team trust increases. With higher levels of trust engendered through improved collaboration, fear is eliminated from the organization (Anderson, 2010)

14 Test Automation

Another improvement suggestion that is considered in the improvement phase of DMAIC for the project under study is to automate test cases.

Test automation is the process of writing a computer program to do testing that would otherwise need to be done manually. Once tests have been automated, they can be run quickly and repeatedly. This is often the most cost effective method for software products that have a long maintenance life, because even minor patches over the life time of the application can cause features to break which were working at an earlier point in time (Hooda, 2012).

14.1 Benefits of Test Automation

Automating test cases has a lot of benefits in agile environment:

- Defects can be found and tracked at an early stage, providing a better insight into the root cause of defects.
- It can discover defects that manual testing cannot find.
- Provides the ability to build a test suite that covers every feature in the software.
- It helps to increase management confidence in the software.
- Automated test executes significantly faster than human users, thus allowing the execution of many test cases that covers most of the application features (if not all) during regression testing.
- The automated test cases can be reused to test different versions of the application and with different configurations or different platforms.
- It reduces the cost as the number of resources for regression test is reduced.
- With increased regression testing coverage, more bugs are prevented from escaping to production which in turn reduces cost radically.
- It can be considered as a safety net for developers especially in cases of necessary code refactoring.

- It can help monitor information that are not visible to a person.

14.2 Principles of Test Automation

With the stated benefits, it is important to keep these principles that were identified by Bach (5) in mind:

1. Test automation cannot duplicate human testers. Human testers, even ones who have no special skills or training, are capable of doing and noticing things that no conceivable test automation can do or notice. It's true that, even with its limitations, automation can have substantial value. But it's usually more productive to think of automation as extending the reach of human testing, rather than replacing it. Effective automation efforts therefore begin with effective thinking about testing.
2. Test automation is more than test execution. There is a lot more to test automation than just a computer running tests, such as:
    - Test generation: tools might create specialized data such as randomized email messages, or populate databases, or generate combinations of parameters that we would like to cover with our tests.
    - System configurations: Tools might preserve or reproduce system parameters, or set systems to a particular state.
    - Simulators: Tools might simulate sub-systems or environmental conditions that are not available for testing.

    - Test execution: Tools might operate the software itself, either simulating a user or working through the GUI, or bypassing the GUI and using an alternative testable interface.
    - Probes: Tools might make visible what would otherwise be invisible to humans. They might statically analyze a product, parse a log file, or monitor system parameters.
    - Test management: Tools might communicate test results, organize test ideas, or present metrics.
3. Test automation is vulnerable to instant obsolescence. Software projects revolve around production code. Test code is not production code. So the priorities of a typical software project allow production code to change even when that breaks test code. This is normal, and generally speaking it's a reasonable, economically justified behavior.
4. Test tools are many and varied. Most people, especially managers, think of test tools as those tools on market that are sold as "test tools". They tend to be quite expensive. But, in fact, almost anything can be a test tool, and many utilities sold for other purposes are especially useful for testing. Some tools are free, some are provided in repositories for developers.
5. Test automation can distract you from good testing. Sometimes the technological focus of test automation can lead to a situation where test automation team can become cut off from the mission of software testing and produce a lot of tools and scripts that might look good, but have a little value in terms of a coherent test strategy that makes sense for the business.

15 Improving Software Quality through Continuous Improvement

Applying continuous improvement to better the quality and productivity of the process is difficult, but it is also paramount to the ongoing success of a mission critical project (Dawson, 1994).

Dawson (1994) mentioned that, a continuous improvement process was applied to software project at Motorola called 'Paperless Integrate Manufactured System' (PIMS). The project started by contracting with a third party after the system needs were defined. However, because the full requirements of the project were not well understood by the users or the developers, the early process life cycle was spiral mode-the goal being to get some capability out onto the manufacturing floor, exercise it, find the problems and determine the real requirements, and then reiterate to build each release.

Once the need for higher software development quality was acknowledged internally, improvements to the process were made in several areas.

- **Better project management:** Specific release planning was continuously performed for PIMS and a balance was made between enhancements that help current and future users and features that affected ongoing product support. As the process grew more formal, a new life cycle model which is the waterfall model was adopted.
- **Configuration management:** The complexity of the project and the product also demanded much more formal program configuration management that was necessary when only a few people were working on a few thousand lines of code. Therefore, the team created both a process and an integrated tool to support configuration management and change tracking.

- **Structured requirements analysis and design:** Design options were iterated and reviewed in early project phases, and complete documentation was maintained. Data flow diagrams, data structure diagrams, and program structure charts were used to capture and communicate analysis and design ideas.
- **Development tools:** To increase the quality and productivity of the process, unique and custom tools had to be developed.
- **Testing and acceptance criteria:** All programming work went through a three-tier testing process. First, a software QA test of each new or modified program unit was performed by a developer other than the author. These tests were based on unit test plans. When all enhancements for a given release were completed, system integration testing is performed to exercise interfaces between all programs. Finally, user testing based on predefined functional test plans was completed to ensure functional and regression integrity.
- **Metrics:** After the process had evolved to a fairly stable level, metrics were needed to better understand the process itself and the improvements being attempted. For example:
    - Released software quality: the total released number of defects per 1000 assembly equivalent lines of code.
    - Customer-found defects: the total number of customer-found defects per 1000 assembly lines of code.
    - Post-release problem report activity: the number of newly opened and total open problems by month.

  - Post-release problem report aging. The mean age of open problems and the mean age of closed problems per month.
  - Cost-to-fix post release problems. The total billed cost spent fixing previously released problems each month (Dawson, 1994).

The benefits drawn from the project:

- Systematic improvements required dedicated support from developers and managers.
- It is harder to change culture-people's thoughts and habits-than it is to change technology.
- A formal development process pays for itself in improved quality and efficiency.
- Metrics are key measuring, understanding, and controlling the development process.
- Any project can benefit from formal management of the basic development process (Dawson, 1994).

16 Impact of CMM on Certain Software Critical Factors

A research was done to examine the impact of the capability maturity model (CMM) on certain critical factors in information systems implementation strategy, software quality and software project performance.

Subramanian et al. (2007) mentioned that in the capability maturity model (CMM), there are five levels. These levels are summarized in table 9.

Process Maturity Level	**Description**
Level 1: Initial	Ad Hoc and occasionally chaotic
Level 2: Repeatable	Basic project management; Process discipline to repeat earlier success
Level 3: Defined	Includes level 2; software processes standardized and integrated; Projects use these approved processes
Level 4: Managed	Includes level 3; Detailed metrics of software product and process are collected; Software process and product controlled using these metrics
Level 5: Optimized	Includes level 4; Continuous process improvements enabled by quantitative feedback; Innovative ideas and technologies developed based on feedback

Table 9: Capability maturity model levels (Sabramanian et al., 2007)

Subramanian et al. (2007) explained four conceptual strategies for information system implementation that were first proposed by Alter (1979).

1. Keep it simple (simplicity): Pressman (2004) mentioned that, functional simplicity (minimum necessary to meet requirements), structural simplicity (modular architecture), and code simplicity (following a coding standard) are key components of simplicity in software. Rajagopal and Frank (2002) explained that complicated, integrated software systems require careful planning before implementation (as cited in Subramanian et al., 2007).

Complex systems inherently present unique risks due to tightly linked interdependencies of business processes, relational databases, and process re-engineering (Subramanian et al., 2007).

2. Executive (or top management) participation and commitment: employee empowerment and executive commitment are two key factors used in Parzinger and Nath, 2000. Powell (1995) mentioned that open organizations, employee empowerment and executive commitment are more critical to the success of total quality management (as cited in Subramanian et al., 2007).
3. Training: Training is a crucial component in continuous improvement. Harel and Tzafrir (1999) mentioned that training also helps to improve employee participation and involvement in quality programs through propagation of priorities and missions of the organization (as cited in Subramanian et al., 2007).
4. Prototyping / evolutionary development: prototyping is recommended for clarity in understanding system requirements and in planning systems architecture (Boehm and Papaccio, 1988) and thus can help in improving software quality (Subramanian et al., 2007).

Information systems (IS) project outcomes

1. Software quality: The quality of software is estimated by many of its attributes such as reliability, integrity, maintainability, enhanceability (extensibility), usability, portability, and reusability (Subramanian et al., 2007). Yang (2001) also pointed

out that the functionality of the software and the appearance of the user interface could affect software quality (as cited in Subramanian et al., 2007).

2. Project performance: In time and within budget are common yardsticks for project performance (Subramanian et al., 2007). Specifically, reducing cycle times and development effort (main factor in software cost and budget) are project performance yardsticks in Harter et al., 2000 (as cited in Subramanian et al., 2007).

In Subramanian et al. (2007) research, they argue that capability maturity model levels influence the choice of IS implementation factors such as training, executive commitment, simplicity, and prototyping which in turn impacts software quality and project performance. They proposed the following hypothesis:

H1: Organizations in different levels of capability maturity model adopt different IS implementation strategies (keep it simple, executive commitment, training, and prototyping) for IS project implementation

H2: IS implementation strategies (keep it simple, executive commitment, training, and prototyping) have a significant impact on IS project outcomes as measured by software quality and project performance.

H3: Organizations in different levels of capability maturity model exhibit different levels of ISproject outcomes as measured by software quality and project performance.

The questionnaires were mailed to 1000 randomly selected IEEE computer society members with an expressed interest in software engineering and a total of 212 responses were received. The questionnaire addressed key practices grouped by:

1) Commitment to perform.
2) Ability to perform
3) Activities performed
4) Measurement and analysis
5) Verifying implementation.

Results

The first hypothesis stated that different capability maturity model levels lead to different IS implementation strategies. The expectation is that higher levels of capability maturity model or process maturity would be associated with different IS implementation strategies. Subramanian et al. (2007) results confirmed the hypothesis.

The second hypothesis studied the effect of IS implementation strategies on IS project outcomes. Here, they had mixed result. While executive commitment and prototyping strategies have significant impact on both software quality and project performance, training had a significant effect only on software quality while "keep it simple" has a significant effect only on project performance. Executive commitment is shown to be a critical factor impacting software quality and project performance (Parzinger and Nath, 2000; Isaac et al., 2004) and is also confirmed by Subramanian et al. (2007) study. Prototyping strategy is expected to impact software quality and project performance based on work by (Boehm, 1988; Boehm and Papaccio, 1988) and Subramanian et al. (2007) study provided empirical confirmation. They also argued that training should not have a significant effect on project completion time, schedule, etc. as training can be easily scheduled in parallel and should not hinder project performance. Training is known to influence software quality (Parzinger and Nath, 2000) and is confirmed to have an effect on quality by Subramanian et al. (2007) study. Keep it simple is a conscious systems design and project management decision that they expected to have a significant effect on project performance and the study confirmed it. Finally, in hypothesis 3, they proposed that different levels of capability maturity model are associated with different IS project outcomes. The expectation was that higher levels of capability maturity model or process maturity would be associated with better software quality and project performance. The result confirmed the hypothesis (Subramanian et al., 2007).

17 Metrics and Management Reporting

The field of software metrics has sufficiently matured so as to allow project managers and software engineers to use metrics to tune software process (Jayanthi & Florence, 2013).

With the use of agile methodology and Kanban tool, Anderson (2010) explained that Kanban's continuous flow system means that we are less interested in reporting on whether a project is "on-time" or whether a specific plan is being followed. What is important is to show: that the Kanban system is predictable and is operating as designed, that the organization exhibits business agility, that there is a focus on flow, and that there is clear development of continuous improvement. We want to track the trend overtime, so we can see the spread of variation. If we are to demonstrate continuous improvement, we want the mean trend to improve over time. If we are to demonstrate improved predictability, we want the spread of variation to decrease and the due-date performance to increase.

Anderson (2010) described some of the metrics that can be used such as:

- <u>Tracking work-in-progress (WIP)</u>

The most fundamental metric should show that the Kanban system is operating properly. To do this, we need a cumulative flow diagram that shows quantities of work-in-progress at each stage in the system. If the Kanban system is flowing correctly, the bands on the chart should be smooth and their height should be stable. An example of cumulative flow diagram is shown in figure 5.

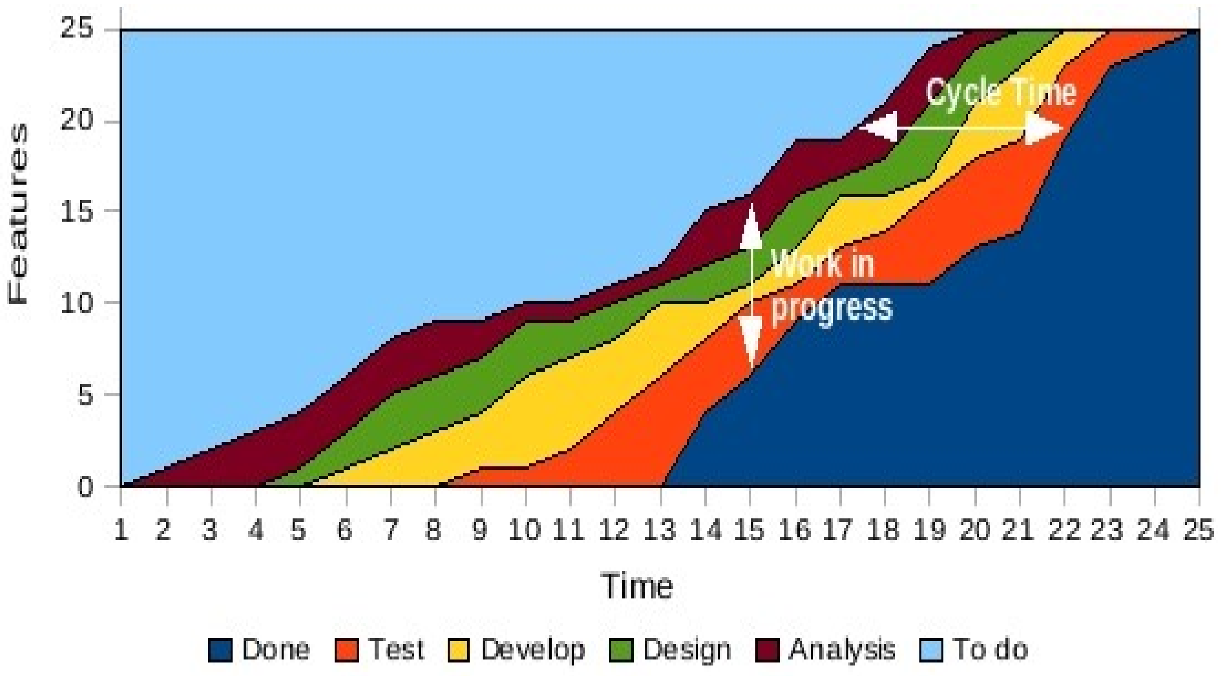

Figure 5: Cumulative flow diagram from a Kanban system

- Lead Time

Lead time can indicate how predictably the organization delivers. If an item was expedited, how quickly did it get from the order into production? If it was of standard class, was it delivered within the target time? Figure 6 shows an example of average lead time.

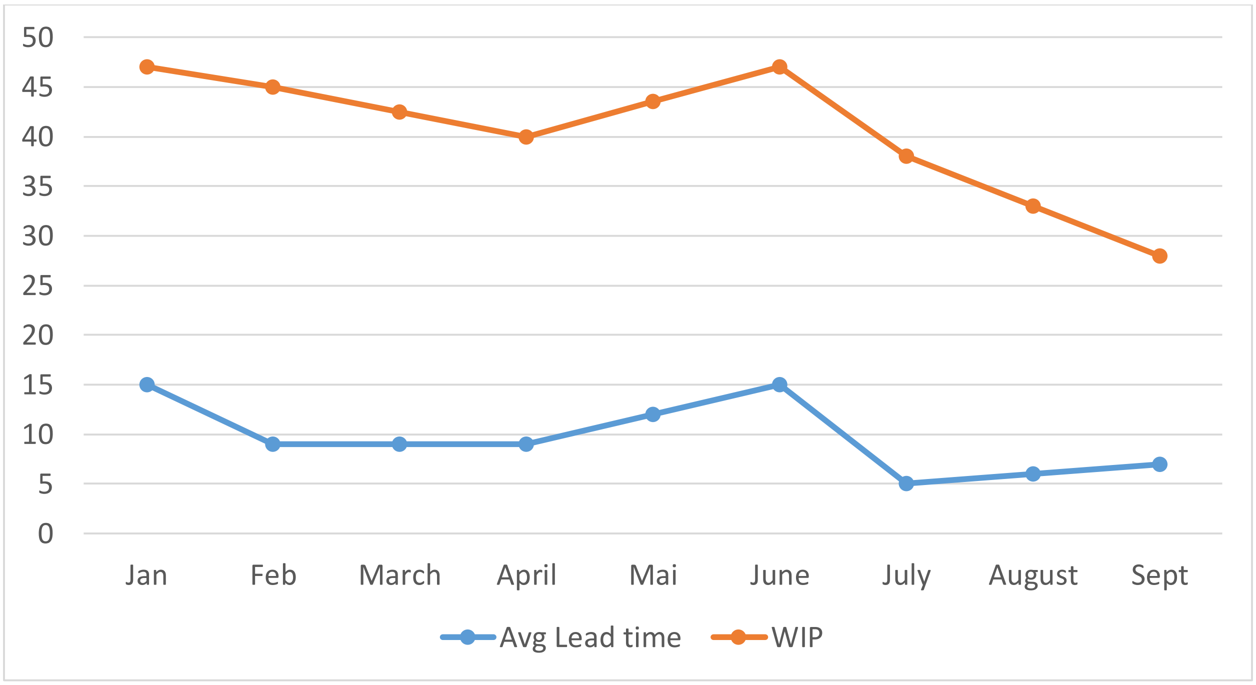

Figure 6: Average Lead Time

- Throughput

Throughput should be reported as the number of items or some indication of their value that were delivered in a given time period, such as one month. Throughput should be reported as a trend over time as shown in figure 7.

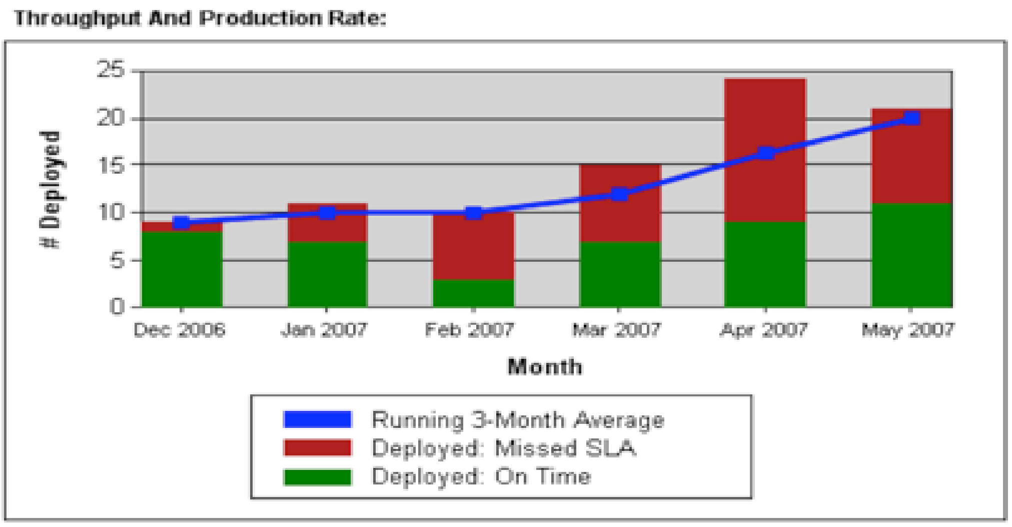

Figure 7: Throughput bar chart (Anderson, 2010)

- <u>Issues and Blocked Worked Items</u>

This chart gives an indication of how well the organization is at identifying, reporting and managing blocking issues and their impact. If due date performance is poor, there should be corresponding evidence in this chart demonstrating that a lot of impediments were discovered and were not resolved quickly enough. This chart can be used on a day to day basis to alert senior management of impediments and their impact. It also can be used as a long term report card to indicate how capable the organization is at resolving impediments and keeping things flowing. It’s a measure of capability in issue management and resolution.

- <u>Initial Quality</u>

Defects represent opportunity cost and affect the lead time and throughput of the Kanban system. It makes sense to report the number of escaped defects as a percentage against the total work-in-progress and throughput. Overtime, we want to see the defect rate fall to close to zero. An example is shown in figure 8.

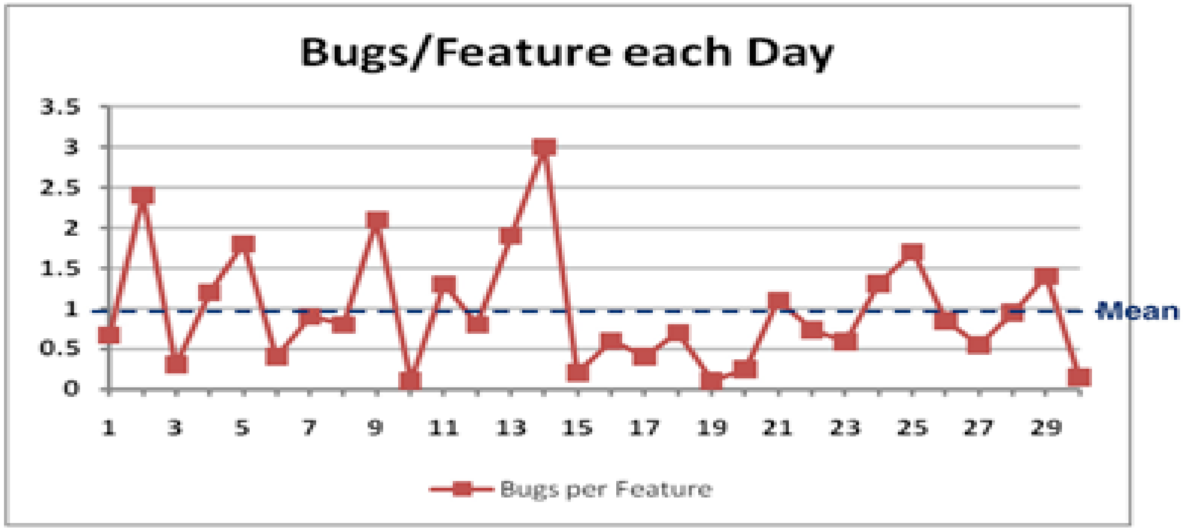

Figure 8: Defects per feature (Anderson, 2010)

Another important measurement tool is the defect removal efficiency.

- Defect removal efficiency (DRE)

Defect removal efficiency provides benefits at both the project and process levels. It is a measure of filtering ability of quality assurance activities as they are applied throughout all process framework activities. It indicates the percentage of software errors found before software release.

It is defined as DRE=E / (E+D)

E is the number of errors found before delivery of the software to the end user.

D is the number of defects found after delivery.

As D increases, DRE decreases (i.e. becomes a smaller and smaller fraction) (Jayanthi & Florence, 2013).

18 Research Gaps

The literature review described a study conducted by Subramanian et al. (2007) that examines the effect of software process maturity (CMM) in the selection of critical information system implementation (IS) strategies and how CMM and the IS implementation strategies impact software quality and project performance. They also mentioned that Card (2004) argued the need for more academic research in software process improvement methods.

Moreover, they mentioned a limitation in their study that it focused primarily on CMM as the software improvement process methodology, and mentioned that other methods such as Total Quality Management, International Standards Organization (ISO) Quality Certification and Six Sigma could also be considered in empirical research.

However, this thesis focuses on using six sigma DMAIC framework as a software improvement methodology that leads to an improvement of the software quality and as a consequence, ensures the success of the software project. The DMAIC framework will be used to identify the root cause(s) of the poor quality for the project under study and helps identify development opportunities that would allow the software organization reach its intended level of excellence through high quality software.

19 Summary and Conclusion

In this chapter a detailed description of six sigma and the importance of the DMAIC framework were provided. Also, software quality factors were identified and the common software quality problems were explained. Most of these challenges come from the traditional software methodology (waterfall) with its failure to cope with the continuous changes required in the current market demand. As a result, agile practices emerged to help overcome those challenges. In this chapter, agile methodology was explained with the focus on its principal values, critical success factors, decision-making challenges and team collaboration. Also, it was mentioned how agile methods can help enhance software quality. The common issues mentioned were also identified in the analysis phase of DMAIC framework in the case study in this paper. As a result, agile methodology was selected as one of the improvement suggestions in the improvement phase of DMAIC based on the enhancements that it provides for the software quality which is the main focus of this study. Other tools such as Kanban and test automation were explained in details as they are used as well in the DMAIC improvement phase for the project under study.

CHAPTER 3:

SOLUTION APPROACH

1 Introduction

In this chapter, we propose a six sigma DMAIC based solution framework to improve software quality in organizations.

2 DMAIC Tools

The DMAIC tools that will be used in the case study are summarized in the table 10.

Phase	Tools Used	Justification of Usage
Define	• Critical to quality (CTQs) • SIPOC	• To determine the metrics that are most important to customers • To establish the boundaries of the business process.
Measure	• Pareto charts	• To prioritize the problem-solving work
Analyze	• 5-Why technique • Cause and effect diagram • Interrelationship diagram	• To isolate the causes from the symptoms • Identify the root cause of the problem • Identify the relationship between different causes
Improve	• Quality function deployment	• To help improve the design phase • To gain client satisfaction
Control	• Measurement metrics	• To help identify whether the improvements applied are going in the right direction • To help identify areas that needs additional focus

Table 10: Summary of Used DMAIC Tools

3 Define Phase

The first phase in the DMAIC process is the define step. The purpose of this phase is to set the project goals bases on the knowledge of the organization, customer critical to quality (CTQ) and the process that needs to be improved.

3.1 Critical To Quality (CTQs)

CTQs are the key measurable indicators of a product or service whose performance standards or specification limits must be met in order to satisfy the customer. CTQs are what the customers expect of a product or service (Chakrabarty & Chuan Tan, 2007).

Defining quality can be a challenge, and it is easy to overlook factors that customers care about. This is when critical to quality (CTQ) are useful. They help to identify and understand quality from customer's point of view, so companies can deliver the service that the customer need and aim for. CTQs align improvement or design efforts with customer requirements (Chakrabarty & Chuan Tan, 2007).

Benefits of CTQs

1. It helps to determine the metrics that are most important to customers.
2. It helps to know what the customer exactly needs and based on that, it will help the organization to focus on those needs as it moves through the process of measuring them and addressing the issues that arise along the way.
3. CTQs help determine the characteristics of the service or product.

3.2 Process Definition - SIPOC Diagrams

SIPOC diagrams are usually used across the DMAIC roadmap for problem solving especially during the define phase. They are a power mapping tool, whose name corresponds to the following five elements: supplier, input, process, output, customer (Marques & Requeijo, 2009).

Moreover, in order to improve the process, it is important to get a high – level understanding of the scope of the current process first. As a result, "SIPOC" is used to identify the boundaries by identifying the process being investigated, its inputs, and outputs, and its suppliers and customers (Evans & Lindsay, 2005).

Benefits of SIPOC

1. It establishes the boundaries of a particular business process.
2. The SIPOC models provide a process-driven approach to divide the entire scope of the six sigma project into manageable partition.

4 Measure Phase

Measure phase is the second step in the DMAIC process. It focuses on measuring the process in order to understand the current performance and as a result manage and systematically improve it.

4.1 Pareto Chart

Pareto analysis refers to the tendency for the bulk of the problems to be due to a few of the possible causes. Hence, by isolating and correcting the major problem areas, obtain the greatest increase in efficiency and effectiveness. The pareto chart is a graphic display that emphasizes the pareto principle using a bar graph in which the bars are arranged in a decreasing magnitude (Koripadu & Subbaiah, 2014).

Benefits of Pareto Chart

1. It is a simple technique for prioritizing problem-solving work.
2. It doesn't only show the most important problem to solve, it also gives a score showing how severe a problem is.

5 Analysis Phase

Analysis is the third phase in DMAIC, it refers to an examination of processes, facts, and data to gain an understanding of why problems occur and where opportunities for improvement exists (Evans & Lindsay, 2005).

Too often we want to jump to a solution without fully understanding the nature of the problem and identify the source or root cause of the problem. Eliminating symptoms of problems usually provides only temporary relief, eliminating root causes provides long term relief (Evans & Lindsay, 2005).

Several approaches can be used to identify the root cause such as the "5 Why" technique, cause and effect diagram and interrelationship diagram.

5.1 5-Why Technique

It is a method of questioning that leads to the identification of the root cause(s) of a problem (Koripadu & Subbaiah, 2014). By asking the question "Why" ideally 5 times, it will allow the analyst to isolate the root cause(s) of the problem from the symptoms. Separating the main issue from the symptoms is crucial as the root cause(s) of a problem get disguised by the symptoms.

Benefits of 5-why technique

1. It is an easy and simple tool to use.
2. It helps to isolate the causes from the symptoms and as a result helps to quickly identify the root cause of a problem.
3. It also helps in defining the relationships between different causes of the problem.

5.2 Cause and Effect Diagram

Cause and effect diagram is a common tool in improvement projects. It is also known as Ishikawa diagram after its originator or as a fishbone diagram. This tool is used to come up with new ideas like in a brainstorming session but in a more balanced way (Kabir et al., 2013). Fishbone diagram was created with the goal of identifying and grouping the causes which generate a quality problem (Ilie & Ciocoiu, 2010).

Benefits of cause and effect diagram

1. (Basic Tools for Process Improvement, 2009) provides a systematic way of looking at effects and the causes that create or contribute to those effects.
2. It helps determine the root causes of a problem or quality characteristics using a structured approach.
3. Encourages group participation.
4. Utilizes group knowledge of the process.
5. Identifies areas where data should be collected for further study (as cited in Ilie & Ciocoiu, 2010).

5.3 Interrelationship Diagram

It is a tool for examining the causes and effect relationship between different factors. It is used to determine which factors have the most impact on other factors. This helps to identify where the effort can be focused to gain greatest benefit. The interrelationship diagram is a special network visualization that consists of a set of nodes connected by arrows. Arrows show directional relationships between "source" (sender) nodes into "target" (receiver) nodes. This representation turns the interrelationship diagram into a form of social network analysis, where connections and interactions between items, objects and systems are made.

Benefits of interrelationship diagram

1. This tool helps gain insights into potential complex relationships of root causes that may underlie recurring problems despite efforts to resolve them.
2. It is useful in prioritizing choices when decision makers find it difficult to reach consensus.
3. It helps in sorting out issues involved in project planning especially when credible data may not exist.
4. It provides a means of evaluating ways in which disparate ideas influence one another.
5. Makes it easy to spot leading factors that affect other factors (Alexander, 2013).

6 Improvement Phase

The goal of six sigma is to accelerate improvements and achieve unprecedented performance levels by focusing on characteristics that are critical to customers and identifying and eliminating causes of errors or defects in processes.

Improve phase is the fourth step in DMAIC and the one that is more difficult to accomplish because it is more of an art than science. While improvement is a highly creative effort, it must be accomplished within the six sigma project management structure (Evans & Lindsay, 2005).

Several improvement suggestions were provided in the case study and one of them is the quality function deployment DMAIC tool.

6.1 Quality Function Deployment

Quality function deployment is a basic product development tool, including design, planning, and communication routines, which provide a methodology directly to relate the customer's needs with engineering characteristics (Thackeray & Van Treeck, 2007). It is a system with the aim of translating and planning the "voice of the customer" into the quality characteristics of products, processes, and services in order to reach customer satisfaction (Bernal et al., 2009).

Benefits of Quality Function Deployment

- Preventive design:

  The biggest advantage of QFD is that it promotes the development of services in a proactive way. When applying QFD, more than 90% of changes on service design are performed before market entry takes place. These changes are much less expensive since they are done at an early stage of the development cycle. This makes it possible to prevent the problems instead of reacting to them (Bernal et al., 2009).
- Reduction of development time:

Having a good design of a software feature that satisfies and exceeds customer requirements, allows a smooth development phase, and thus a reduction of development time as a consequence.

It helps assure that testing verifies conformance to the customer's requirements by providing testable items in the functional specification. This means less reworking of the design and implementation to meet customer requirements. Less reworking means shorter development schedule and reduced development costs. Testing conformance to customer requirements allows test suite to be designed and implemented in parallel with the product design and implementation (Thackeray & Van Treeck, 2007).

- Client satisfaction:

QFD's is oriented to the "voice of the customer" and not to the "thoughts of the developer". With the focus on the consumer, all decisions made during the service design are targeted at the customer (Bernal et al., 2009).

- Take politics out of decisions:

QFD helps take the politics out of decisions. QFD's triangular matrices show the impact of one requirement on other requirements, or one function on other functions, or one design element on other elements, etc. It helps the user and all members of development team to understand better inconsistencies and tradeoffs and also to achieve consensus, which is very important to the success of large projects (Thackeray & Van Treeck, 2007).

7 Control Phase

Control phase is the last step of the six sigma DMAIC process, and is the activity of ensuring that project improvements will be sustained by tracking key performance measurements and CTQs.

This requires monitoring the process and results, and taking corrective action when necessary to correct problems and bring the process back to stable performance. Control is important for two reasons. First, it is the basis for effective daily management of work at all levels of an organization. Second, long – term improvements cannot be made to a process unless the process is first brought under control (Evans & Lindsay, 2005).

7.1 Control System Components

Evans and Lindsay (2005) argued that any control system has three components:

1. A standard or goal.
2. A means of measuring accomplishments.
3. Comparison of actual results with the standard, along with feedback to form the basis for corrective action.

Goals and standards establish what is supposed to be accomplished. These goals and standards are reflected by measurable quality characteristics, such as number of defects or customer complaints.

Measurements supply the information concerning what has actually been accomplished. Workers, supervisors, or managers then assess whether the actual results meet the goals and standards.

Short-term corrective action generally should be taken by those who own the process and are responsible for doing the work. While long-term remedial action is the responsibility of management. Process owners must have the means of knowing what is expected (the standard or goal) through clear instructions and specifications, they must have the means of determining their actual performance, typically through inspection and measurement, and they must have a means of making corrections if they discover a variance between what is expected of them and their actual performance (Evans & Lindsay, 2005).

Several metrics and reports that were described in the literature review can help to make the necessary measurement in order to take the required corrective actions.

CHAPTER 4:

CASE STUDY AT RK COMPANY

1 Introduction

A case study was done at ‘RK’ company by applying DMAIC methodology to one of its largest projects.

2 ‘RK’ Company and DMAIC

‘RK’ Company is one of the fast growing companies in Canada. ‘RK’ Company is a medium size company whose main focus is building software applications using the latest technologies.

DMAIC framework is applied to one of the biggest and core projects for ‘RK’ company. This project has been in development for the past few years and the current followed methodology is waterfall. The development team is actually divided into six sub-teams, where each team is specialized in developing a specific area in the application. In total the project has 35-40 developers. There is also a system test team (or quality assurance team) that consist of 12-14 system test engineers that test the product manually. There is also another team which is the software developer engineers in test team that should create automated test cases for the project. This team was recently added and their current focus is to create test tools for the project.

The emphasis is on applying DMAIC framework on the described project in order to identify the root causes for the large number of bugs found in production and provide improvement suggestions for ‘RK’ Company and help the company to control its process to ensure the success of the project.

For the purpose of this study, interviews were carried out with the software development manager and the quality assurance manager in order to collect information about the process followed and the issues the team has. Also, Production bugs were collected from the company reporting system and were classified by type, severity and seasonality. The collected information will be used throughout the DMAIC phases.

3 Define Phase

In this phase the CTQs of the project under study are identified and the boundaries of the current process were defined through the use of the SIPOC diagram.

3.1 Critical To Quality (CTQs)

Based on the benefits of CTQs identified in the solution approach, CTQs are used for the project under study in order to determine the quality that is expected by the customers of the project.

The most important quality attributes that contribute to customer perceptions for the project under study are the following:

- **Security:** Security relates to the ability of software to prevent prohibited access and withstand deliberate attacks intended to gain unauthorized access to confidential information, or to make unauthorized access (Chang et al., 2006).

  Since the project under study is based on security concept, this attribute is the main key for customers, thus the security of the software must be assured in order to guarantee the business continuity of the product. The design and implementation of the software should protect the data and resources contained and controlled by that project.

- **Reliability:** Quyoum et al. (2010) explained that software reliability is an important facet of software quality. Software reliability is the probability of the failure free operation of a computer program for a specified period of time in a specified environment. Unreliability of any product comes due to the failures or presence of faults in the system. Thus, the unreliability of software is primarily due to bugs or design faults in the software.

- **Assurance:** This is related to the knowledge and courtesy of employees, and their ability to convey trust and confidence (Evans & Lindsay, 2005). For example, the ability of the project support team to answer questions, have the capabilities to do the necessary work to fix customer issues in a timely manner and be polite and pleasant during this time.
- **Efficiency:** Chang et al. (2006) explains that it relates to how the software optimally uses system resources. It includes the time behavior as the ability of software to provide appropriate responses, processing time and throughput rate when performing its function under stated conditions. It also includes resource behavior which is the ability of software to use appropriate resources in time when the software implements its function under stated conditions.
- **Maintainability:** Which refers to the ease with which a software can be understood, modified and retested. The easier the software can be maintained, the easier it is to isolate defects or their causes, correct defects or their causes, maximize the software useful life, maximize its efficiency, reliability and safety as well as meet new requirements and cope with a changed environment. This key attribute is not properly applied to the project under study. The code of the software was not designed to be testable. Also, the complexity of the code does not help in isolating defects or correct them.

3.2 Process Definition - SIPOC Diagrams

The below “SIPOC” (figure 9) identifies the software testing process that is currently implemented for the project under study.

Supplier	Input	Process	Output	Customer
Organization	Test Manager Tool	Create manual test cases for new planned features	New feature test suite	Project manager
QA team	High level requirements document (HLR)	Execute test cases for each fully implemented new feature	New feature test results	Development manager
QA manager	System requirements document	Report bugs for new features	List of identified bugs for new feature	QA manager
Development team	Project stable build	Verify bug fixes for new features	Modified build after bug fixes	Organization
Development manager	Testing Tools	Retest all new features when all new planned features are fully implemented.	Regression test results	
Project manager	Testing environment	Perform regression testing	Updated test cases	
	Test plan	Report bugs found in regression	Updated regression test suite to include priority one test cases of newly implemented features	
	Regression test suite	Verify bug fixes for regression testing		
	Legacy system knowledge	Perform manual test case maintenance (clean up)		

Figure 9: Project SIPOC Diagram

4 Measure Phase

The goal is to reduce the number of bugs found in production as low as possible and to be able to find them as early as possible in the iteration cycle. As a result the cost will be radically reduced.

In order to generate a useful process performance measure for the project under study, the following points were taken into consideration:

1. Customer requirements and expectations that were identified in the critical to quality (CTQs) in the define phase.
2. Work process definition that provides the service which was also identified in the define phase.
3. Develop specific performance measures or indicators which are based on the stated goal.

Concentrating on the stated goal, the data collection focused on the number of bugs found in production for the past releases in 2012, 2013 and 2014 and was identified by severity and the type of issues found. It focused on the population in order to achieve more accurate results. Several Pareto charts were created to summarize and display the relative importance of the differences between groups of data such as the seasonality of releases, severity of bugs found and the type of errors identified in production. The Pareto charts help to identify which situations are more significant and as a result will help to know where to direct the improvement efforts.

The following sections shows different Pareto charts for the project under study.

4.1 Pareto Chart Based on the Type of Errors

The issues found in production were classified by type such as functional, backend services, help information, graphical user interface, run time error, translation, scalability and configuration.

This below chart was based on the type of errors.

Types Of Errors Identified	Number of bugs	cumulative number of bugs	Cumulative Percentage
Functional	477	477	44%
Backend Services	357	834	76%
Help Information	105	939	86%
Graphical User Interface	95	1034	94%
Runtime error	27	1061	97%
Translation	15	1076	98%
Scalability	10	1086	99%
Configuration	9	1095	100%

Table 11: Bugs per type

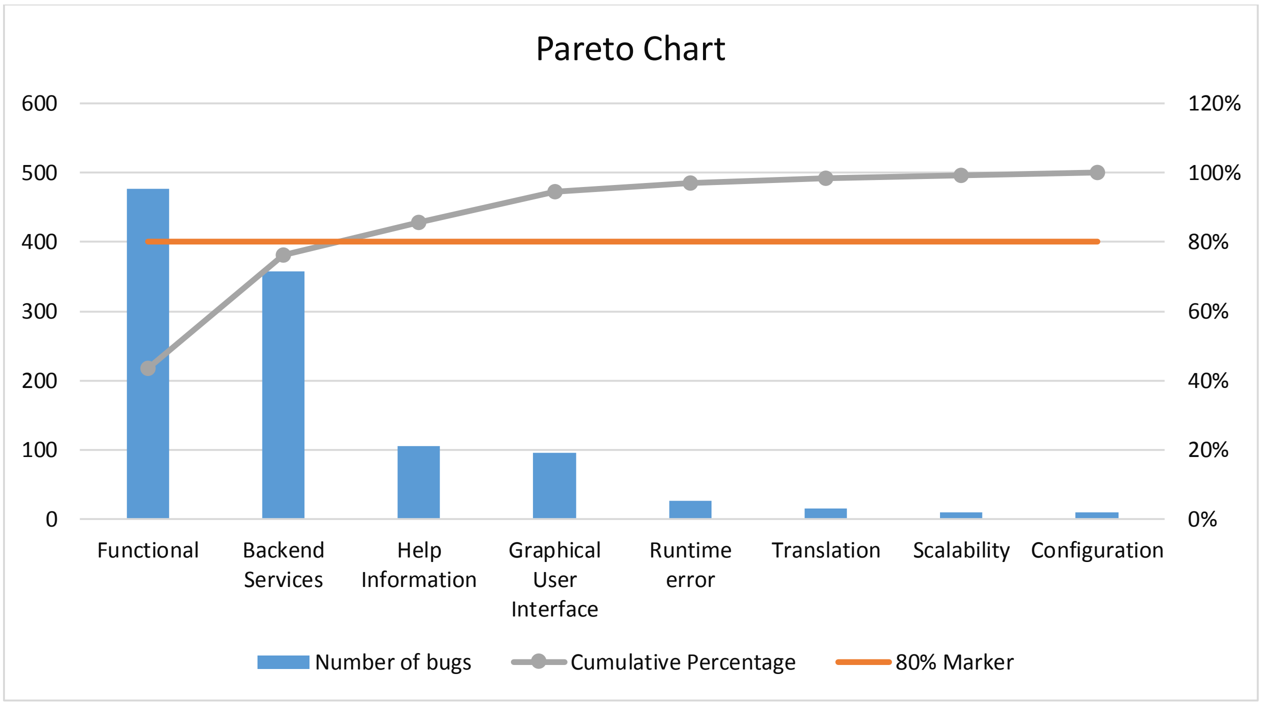

Figure 10: Pareto Chart based on type of errors

From this chart we can conclude that 'RK' company needs to focus on improving the functional and backend services.

4.2 Pareto Chart Based on the Seasonality

The faults found in production were also classified by year and as a result the below chart identifies the errors found per year.

Year	Number of Bugs	Cumulative Number of Bugs	Cumulative Percentage
2013	640	640	58%
2014	250	890	81%
2012	205	1095	100%

Table 12: Bugs per seasonality

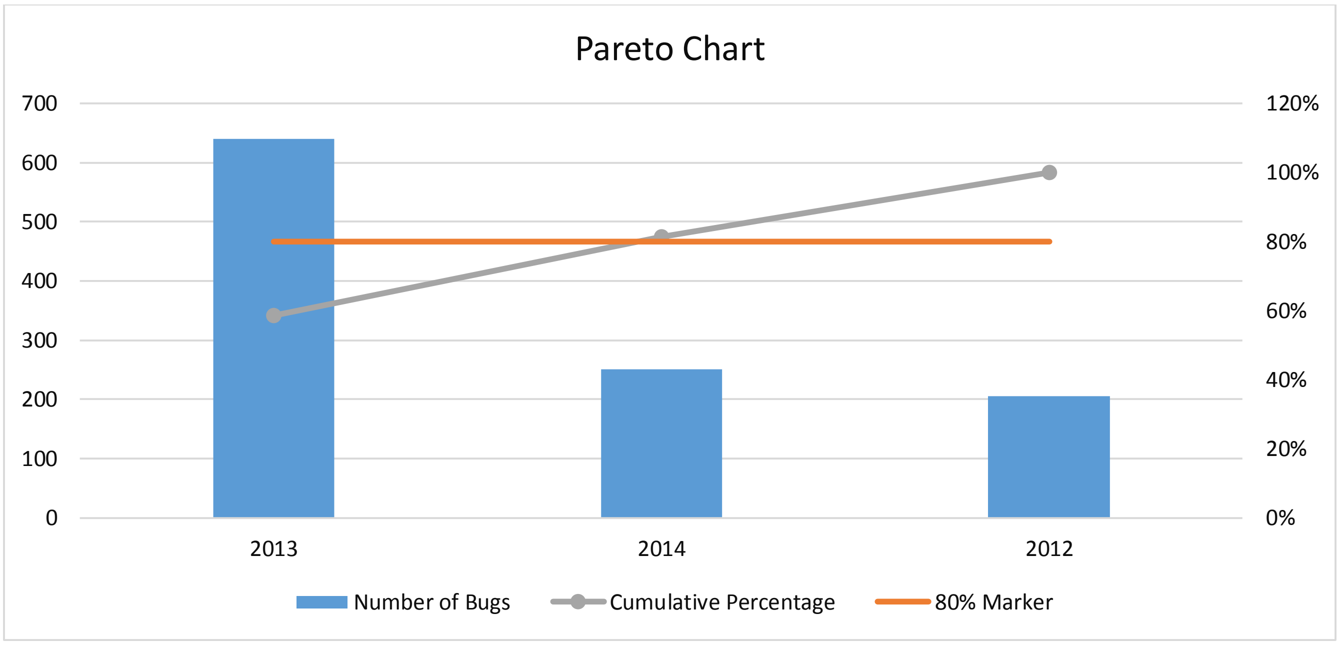

Figure 11: Pareto chart based on seasonality

The above Pareto chart shows that ‘RK’ company needs to concentrate their testing and bug fixes on the features that got released and the affected functionalities in these two years.

4.3 Pareto Chart Based on the Severity of the Bugs

The bugs found in production were also classified by severity and as a result the below chart focuses on this classification.

Severity	Number of Bugs	Cumulative Number of Bugs	Cumulative Percentage
Medium	631	631	58%
Major	272	903	82%
Minor	170	1073	98%
Critical	22	1095	100%

Table 13: Bugs per severity

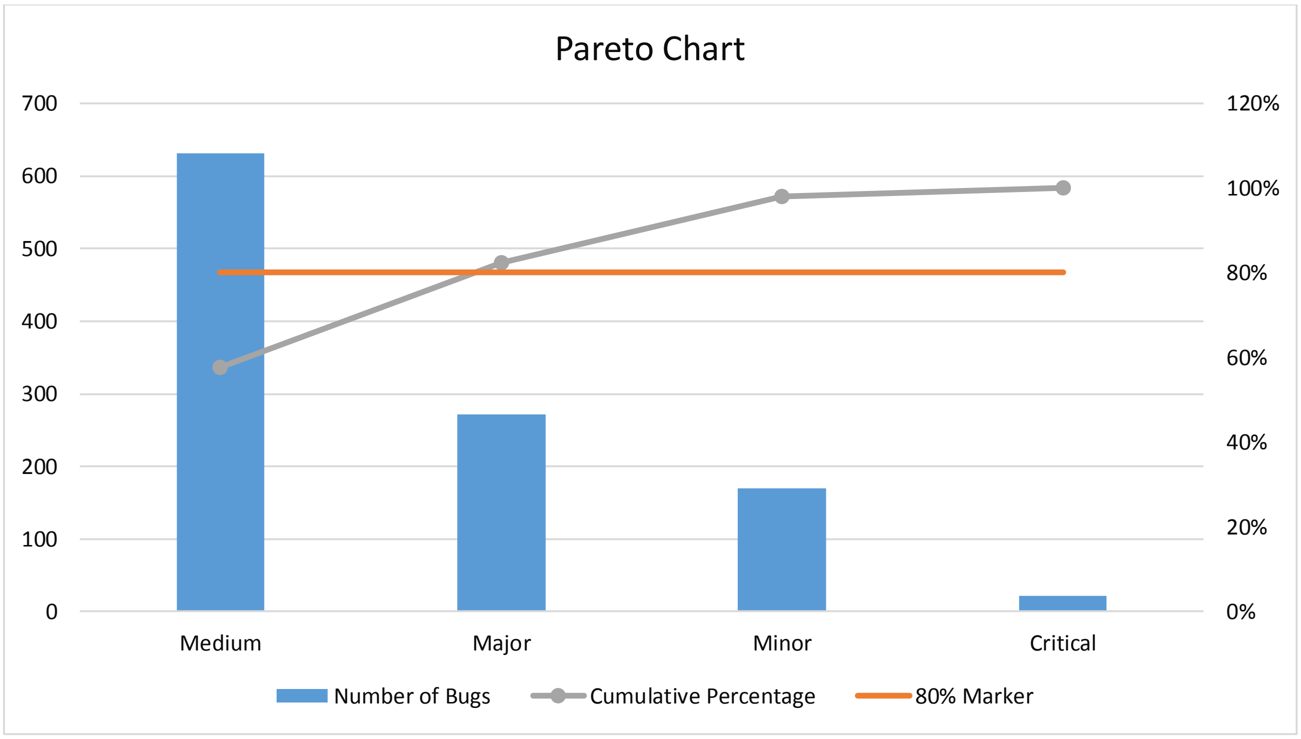

Figure 12: Pareto chart based on severity

Looking into the above Pareto chart based on the severity of the bugs, we notice that medium and major bugs have the highest ratio.

4.4 Pareto Chart Based on the Type of Errors and Severity

The below chart was prepared based on the type of bugs and their severity.

Type of errors and severity	Number of Bugs	Cumulative Number of Bugs	Cumulative Percentage
Functional Medium	281	281	26%
Backend Medium	199	480	44%
Backend Major	121	601	55%
Functional Major	113	714	65%
Help Information Medium	74	788	72%

Functional Minor	65	853	78%
GUI Medium	45	898	82%
GUI Minor	41	939	86%
Backend Minor	33	972	89%
Help Information Minor	21	993	91%
Functional Critical	18	1011	92%
Runtime Medium	15	1026	94%
Help Information Major	10	1036	95%
Runtime Major	10	1046	96%
GUI Major	9	1055	96%
Translation Medium	8	1063	97%
Translation Minor	6	1069	98%
Scalability Major	6	1075	98%
Configuration Medium	5	1080	99%
Backend Critical	4	1084	99%
Scalability Medium	4	1088	99%
Runtime Minor	2	1090	100%
Configuration Major	2	1092	100%
Configuration Minor	2	1094	100%
Translation Major	1	1095	100%

Table 14: Bugs per type and severity

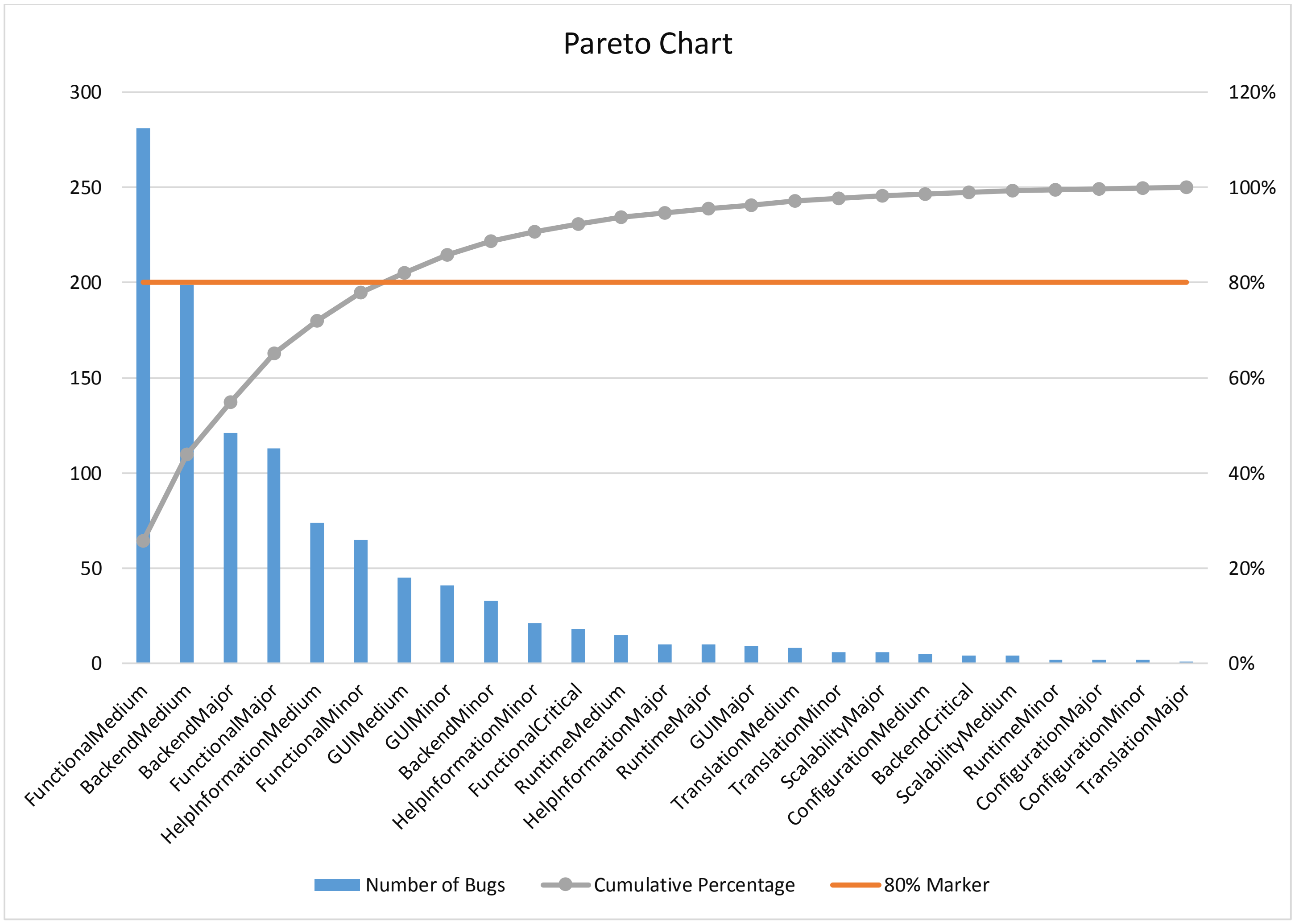

Figure 13: Pareto chart based on type of errors and severity

The above Pareto chart shows that major and medium functional and backend services bugs should have the main focus.

4.5 Pareto Chart Based on the Severity and Seasonality

The below chart was prepared based on the severity of the bugs and the year they were identified.

Severity and Seasonality	Number of Bugs	Cumulative Number of Bugs	Cumulative Percentage
Medium 2013	360	360	33%
Major 2013	169	529	48%
Medium 2014	149	678	62%
Medium 2012	122	800	73%
Minor 2013	94	894	82%
Major 2014	53	947	86%
Major 2012	50	997	91%
Minor 2014	44	1041	95%
Minor 2012	32	1073	98%
Critical 2013	17	1090	100%
Critical 2014	4	1094	100%
Critical 2012	1	1095	100%

Table 15: Bugs per severity and seasonality

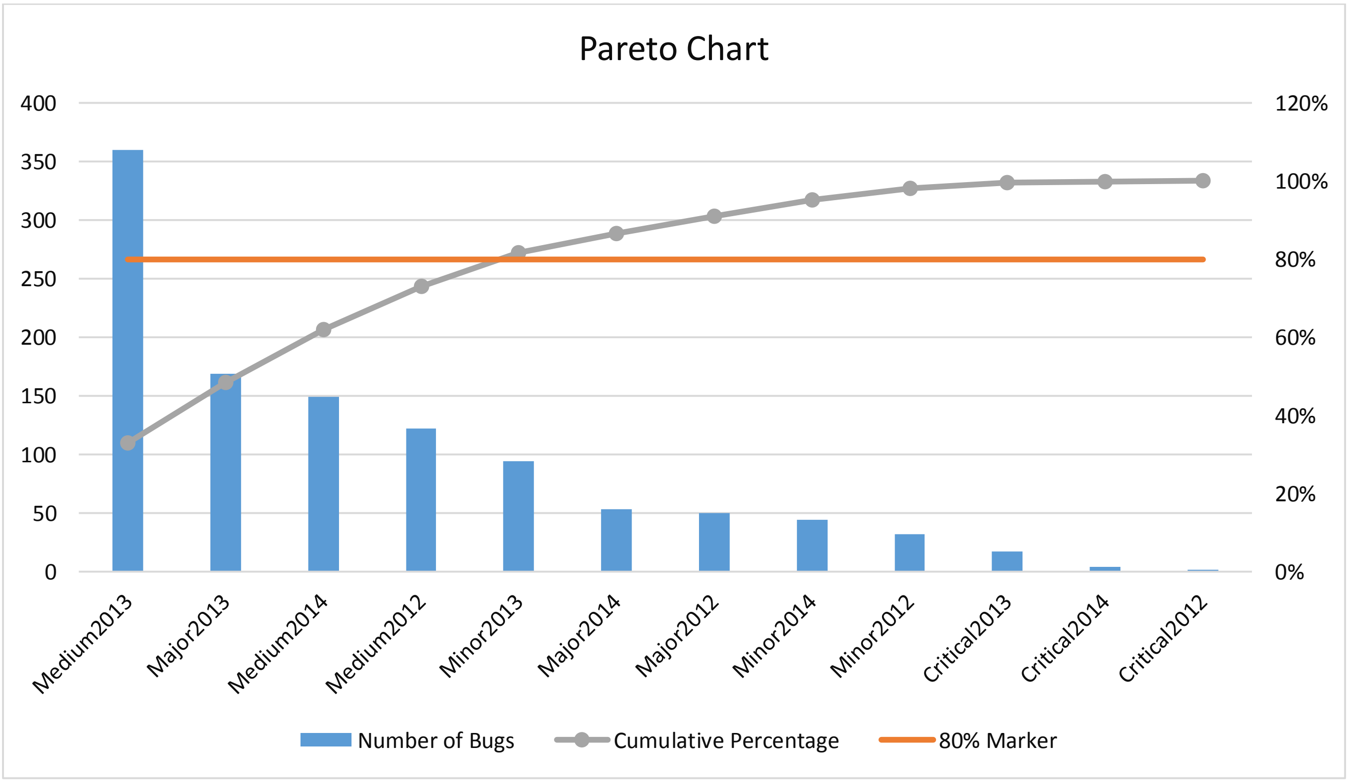

Figure 14: Pareto chart based on severity and seasonality

This Pareto chart expresses that ‘RK’ company should focus on medium and major 2013 bugs as well as medium bugs for 2014 and 2012.

4.6 Pareto Chart Based on the Type of Errors per Season

The below chart was prepared based on the severity of the bugs and the year they were identified.

Type of errors per year	Number of Bugs	Cumulative Number of Bugs	Cumulative Percentage
Functional 2013	270	270	25%
Backend 2013	211	481	44%
Functional 2014	112	593	54%
Functional 2012	95	688	63%
Backend 2014	75	763	70%
Backend 2012	71	834	76%
Help Information 2013	66	900	82%
GUI 2013	57	957	87%
Help Information 2014	30	987	90%
GUI 2014	20	1007	92%
Runtime Error 2013	19	1026	94%
GUI 2012	18	1044	95%
Help Information 2012	9	1053	96%
Translation 2013	7	1060	97%
Translation 2012	6	1066	97%
Scalability 2013	6	1072	98%
Runtime Error 2014	5	1077	98%
Configuration 2013	4	1081	99%
Scalability 2014	4	1085	99%
Configuration 2012	3	1088	99%
Runtime Error 2012	3	1091	100%
Configuration 2014	2	1093	100%
Translation 2014	2	1095	100%

Table 16: Bugs per type and seasonality

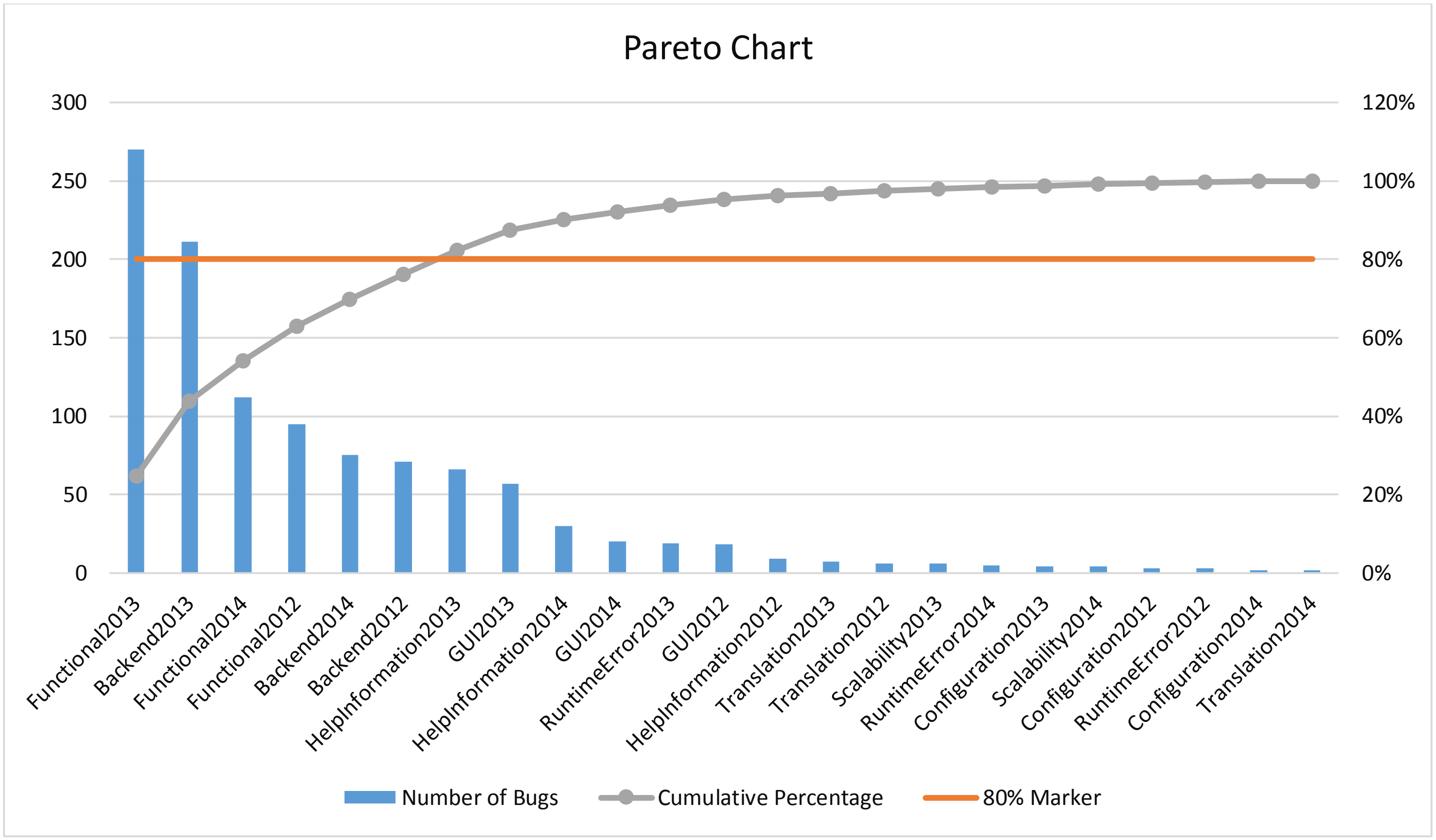

Figure 15: Pareto chart based on type of errors per season

Applying the 80:20 rule based on the type of errors per season displays that 'RK' company should focus on functional 2013, 2014 and 2012 bugs as well as the backend bugs in 2012, 2013 and 2014.

5 Analysis Phase

In this phase we analyzed the results obtained from the measurement phase. Also, several techniques were used to identify the root cause of the large number of bugs found in production such as the 5-Why technique, cause and effect diagram and interrelationship diagram.

5.1 Analysis Results

In the measure phase, we applied the 80:20 rule. The Pareto principle (or 80:20 rule) states that most effort (approximately 80%) is due to a limited number of key actions (approximately 20%) (Gentleman et al., 2012). This principle is also called "vital few and trivial many".

Analyzing the collected data in the measure phase and applying the Pareto chart helps to identify the "vital few" from trivial many in order to identify direction for selecting the areas that need more intensive focus to improve the quality of software.

- In the Pareto chart based on the type of errors (Figure: 10) and by applying the 80:20 rule we notice that 'RK' company needs to focus on fixing the functional and backend services issues.
- In the Pareto chart that is based on seasonality (Figure: 11), we notice that the year 2013 had the highest number of bugs followed by 2014. This shows that 'RK' company needs to concentrate their testing and bug fixes on the features that got released and the affected functionalities in these two years.
- Looking into the Pareto chart based on the severity of the bugs (Figure: 12), we notice that medium and major bugs have the highest ratio. However, this result does not imply that the focus should not be on critical issues as well since this type of severity should not even be found in production and could be showstoppers. Critical bugs do not have work around and should be addressed immediately.
- Analyzing the collected data and the Pareto chart (Figure: 13) based on the type of errors and severity also shows that major and medium functional and backend services bugs should have the main focus.

- As for the Pareto chart based on the severity and seasonality (Figure: 14) illustrates that company 'RK' should centralize their attention on medium and major 2013 bugs as well as medium bugs for 2014 and 2012.
- Applying the 80:20 rule based on the type of errors per season (Figure: 15) shows that functional 2013, 2014 and 2012 bugs should have company 'RK' attention as well as the backend bugs in 2012, 2013 and 2014.

As a result, we can conclude that functional and backend services need more intensive testing in order to improve the quality of the project. However, we should look more into finding the causes that led to this number of bugs found in production.

5.2 5-Why Technique

5-why approach forces one to redefine a problem, statement as a chain of causes and effects to identify the sources of the symptoms and as a result it is applied to the project under study:

1) Why there is too many bugs found in production in each release?

   This is because it wasn't tested properly.
2) Why it wasn't tested properly?

   This is due to the fact that many regression and performance tests were cut in many cases.
3) Why test cases were cut?

   Because the test schedule is not adequate for the number of tests that should be performed.
4) Why there wasn't enough time to execute all necessary tests?

Because developers never deliver on time and there is a large regression on legacy features that need to be executed.

5) Why developers never deliver on time?

Because of the many features that are added at each release and in some cases client requirements or new features are added or UI changes are done in the middle of implementation leading to a change in the scope and the current methodology does not properly adapt to the requirements change.

5.3 Cause and Effect Diagram

Based on the stated benefits in the solution approach, the cause and effect diagram is applied to the project under study to group together the issues identified from the interviews conducted with the QA manager and the development manager for the project under study. It allowed us to identify the root cause of the problem.

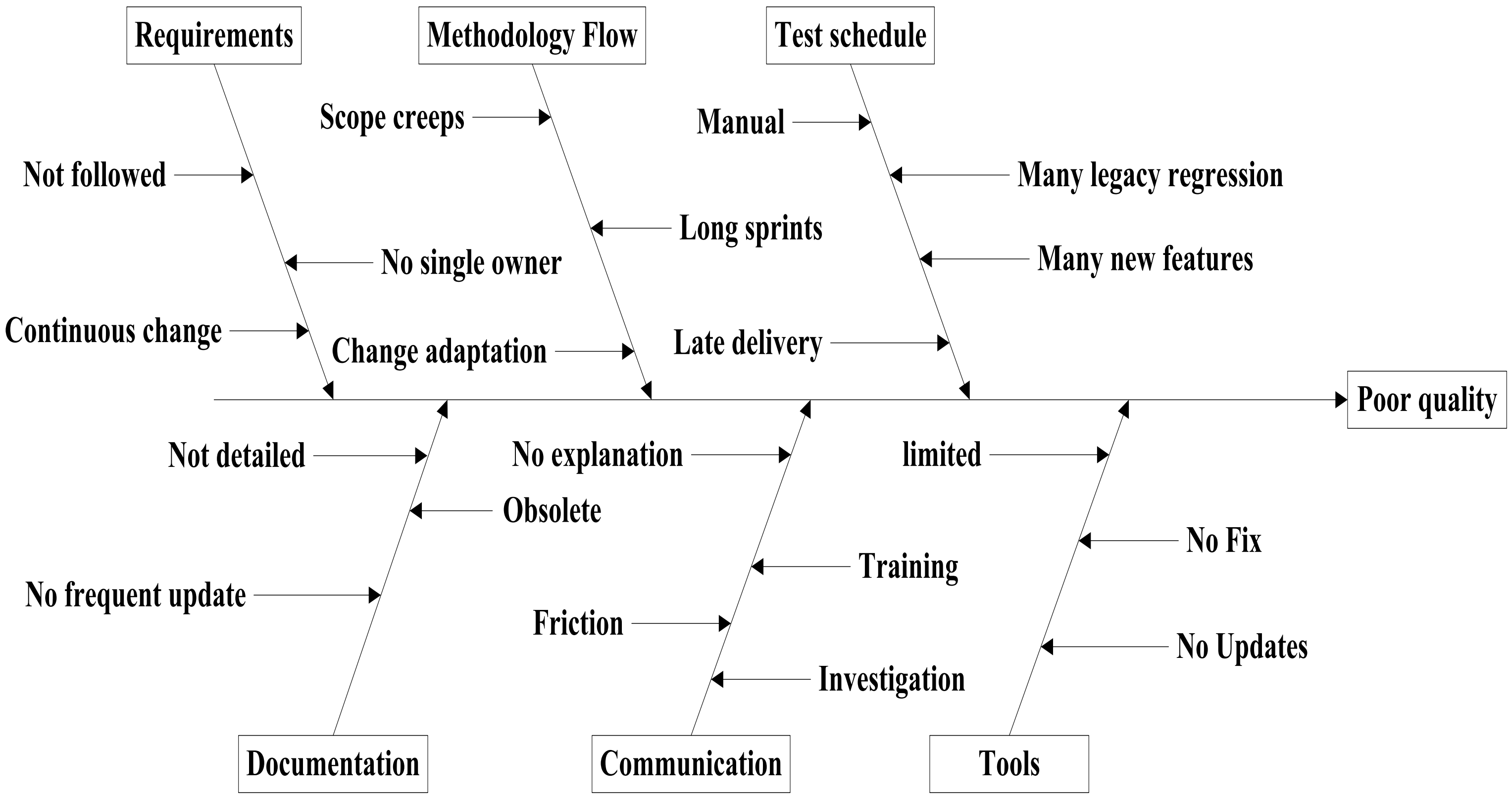

Figure 16: Cause and effect diagram

5.4 Interrelationship Diagram

The interrelationship diagram is applied for the project under study as shown in figure 17 in order to identify the potential causal relationships that might lie behind a problem that continues to recur despite attempts to resolve it.

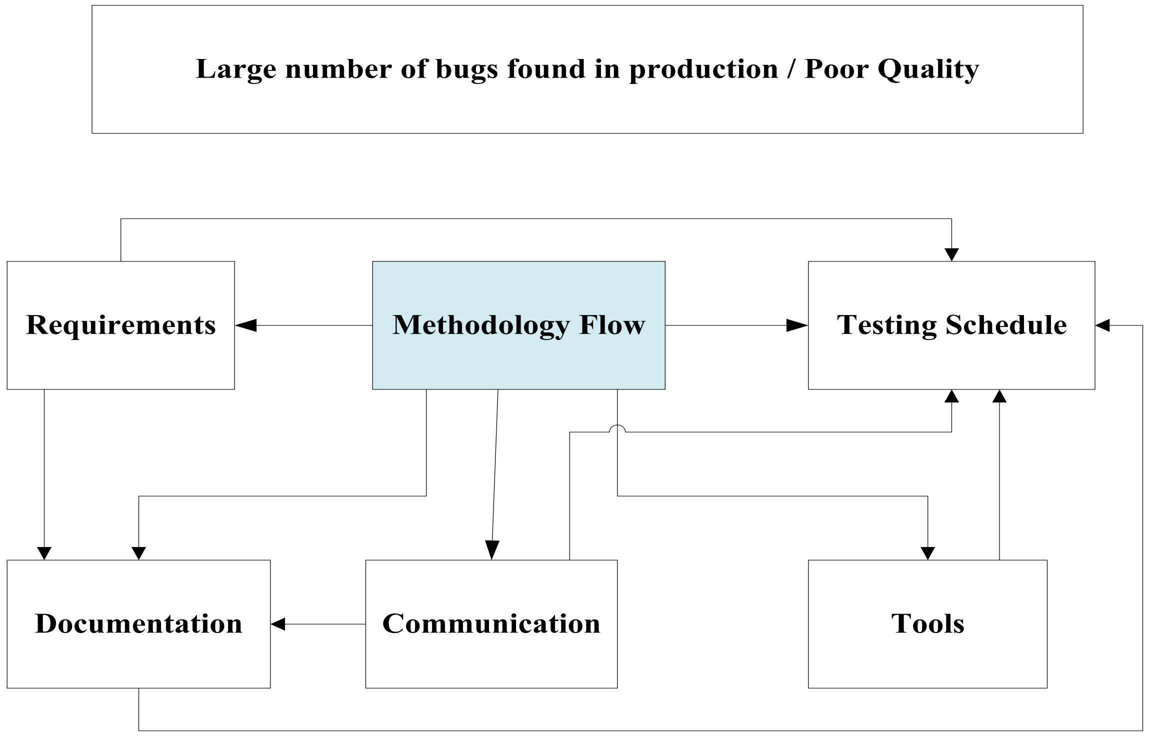

Figure 17: Interrelationship diagram

From the interrelationship diagram we can conclude that the currently methodology followed is the root cause of poor software quality for the project under study.

5.5 Analysis Conclusion

The analysis phase of the project under study revealed that the areas that need intensive testing are the functional and the backend services as they have the largest number of bugs found in production in the analyzed years. However, test coverage is not adequate due to insufficient testing schedule.

Several analysis tools such as the “5 why” technique, cause and effect diagram and the interrelationship diagram are used to separate the symptoms from the causes and to identify the root cause of the problem. We noticed that there are several symptoms that helped to uncover the main cause. Developers rarely deliver to the system test team with a stable build to work with as scheduled. The sprint is too long and there are too many features that get planned to be released at a time. Also, when system test team start reporting bugs, developers have already started new implementations and have forgotten what and how they implemented the features under test. As a result, it takes them a lot of delay time to debug and figure out the problem. In consequence, it takes a long time to have a functioning feature.

Also, some client requirements or new features are added or user interface changes are made in the middle of implementation where the current used methodology fails to cope with. The description of the waterfall model flow and its disadvantages described in the literature review chapter explain the above mentioned issues. Thus leading to the discovery that the current waterfall methodology used is the main cause of the large number of bugs in production.

Also, from the advantages and disadvantages of waterfall methodology, it was mentioned that waterfall model is not suitable for moderate to large projects which is the case of the project under study. Also, waterfall is not a good model for complex and object oriented projects which is also the case of the project under study. 'RK' Company project includes different modules developed by several separate teams that get integrated together.

Another mentioned disadvantage of waterfall model is that it is not suitable for projects where requirements change all the time which leads to a high risk.

6 Improvement Phase

In this phase, several improvement suggestions were provided for the project under study in order to achieve high quality product.

6.1 Improvement Suggestions

Six sigma project selection focuses on improvement opportunities that have a verifiable financial return. Such opportunities include the obvious reductions in production defects (Evans & Lindsay, 2005). This will be my focus and the goal of the coming improvement suggestions.

The project under study needs to improve the project flexibility, reduce the cycle time and make the process flow continuously.

- **Flexibility:** The suggested flexibility refers to the ability to adapt quickly and effectively to changing requirements (Evans & Lindsay, 2005), which is one of the reasons identified in the analysis phase of the project under study. It refers to the ability to respond rapidly to changing demands and the ability to produce a wide range of customized services.

  To be able to succeed in globally competitive markets, requires a capacity for rapid change and flexibility (Evans & Lindsay, 2005).
- **Cycle time:** Refers to the time it takes to accomplish one cycle of a process. Reductions in cycle time serve two purposes. First, they speed up work processes so that customer response is improved. Second, reductions in cycle time can only be accomplished by streamlining and simplifying processes to eliminate non-value added steps. This approach forces improvements in quality by reducing the potentials for mistakes and errors. By reducing non-value added steps, costs are reduced as well. Thus, cycle time reductions often drive simultaneous improvements in organization, quality, cost and productivity (Evans & Lindsay, 2005).

Flexibility and cycle time are the pillars for agility which is crucial to such customer – focused strategies as mass customization, which requires rapid response and flexibility to changing customer demand.

Thus, this leads to the main suggestion for ‘RK’ company to move from waterfall process to “Agile Methodology” to improve the required quality that ensures the success of the project.

However, there are several popular agile methods such as Extreme Programming and Scrum. The suggested method to be used by ‘RK’ company for the project under study is “Scrum” based on its ability to split a large team into smaller sub-teams which is already the case of the project under study.

6.2 How Can ‘RK’ Company Implement Scrum Method

Schwaber and Beedle (2002) suggested that the team should consist of five to nine members and if more people are available, several sub-teams can be formed (as cited in Abrahamsson et al. 2002). This is already the case of the project under study. The project team is already formed of six sub-teams, which facilitates the switch from waterfall methodology to agile methodology using scrum method and reduces resistance to change.

Following Schwaber and Beedle (2002) suggestion that was mentioned in (Abrahamsson et al. 2002), ‘RK’ Company will be adopting scrum for an existing project where the development environment and technology to be used exists, but the project team has problems related to complex technology and changing requirements that were identified in the analysis phase. In this situation, scrum should be started with daily standup meetings. The objective of the first sprint should be to demonstrate any piece of user functionality on the selected technology. The team should work on solving the impediments of the project that will enable the team to progress.

Focusing on the description of scrum practices given by Schwaber and Beedle (2002), 'RK' company needs to define everything that is needed in the final product based on current knowledge in the product Backlog. The Backlog list should be constantly updated with the list of requirements. It can also include features, functions, bug fixes, defects, enhancements or technology updates. This list should be controlled by adding updating and removing work items. The product owner is responsible for keeping the product backlog up to date (as cited in Abrahamsson et al., 2002).

The team should apply sprint procedure in order to adapt to changing environmental variables. Every sprint should begin with the sprint planning meeting in which the product owner and the team discuss which stories will be moved from the product backlog into the sprint backlog.

During the sprint, the team should have a daily scrum meeting for fifteen minutes during which the team discuss solutions to challenges and report progress to the product owner. At the end of the sprint, a review meeting should be held to present the work done to the product owner in order to determine if the work done has met its acceptance criteria.

Also, the team should have a retrospective meeting to discuss what went good and what needs to be improved and provide improvement suggestions.

After stabilizing the change of moving from waterfall to agile using scrum method, 'RK' company can work on optimizing the process by looking into other tools for directly improving service delivery and catalyzing continuous improvement. This tool is called "Kanban". Using Kanban will help the project under study to maintain a constant pace indefinitely.

6.3 How Can 'RK' Company Implement Kanban

'RK' Company can benefit from Kanban system in addition to the suggested agile scrum method to limit the team's work in progress to set capacity and to balance the demand on the team against the throughput of their delivered work. By doing this they can achieve sustainable pace.

Since each software team has different situation, the team of the project under study need to experiment or in another way evolve their process to best suit their needs. Since the team members are capable of understanding the basic principles of scrum and Kanban, they are therefore capable of inspecting, tailoring and adapting to the process that fit their context and optimize it to their domain.

Following the steps that Anderson (2010) defined in his book:

1. First 'RK' company needs to define the start and end point for control. It is necessary to decide where to start and end process visualization, and in doing so, define the interface points with upstream and downstream partners. The team for the project under study can adopt workflow visualization with cards and limit work-in-progress within their own political sphere of control and negotiate a new way of interacting with immediate upstream and downstream partners. For example, the development manager and test manager who have control over the analysis, design, testing and coding, can map this value stream and negotiate new styles of interaction with the business partners upstream who provide requirements, prioritization and portfolio management and those downstream with system operations.
2. The next step is to identify the types of work that arrive at that point and any others that exit within the workflow that will need to be limited. For example, bugs are likely a type of work that exists within the workflow. The team can also identify other types such as code

refactoring, system maintenance and upgrades. For incoming work, the team can have types like user stories.

3. The third step is to draw cards wall to show the activities that happen to the work rather than specific functions or job descriptions. Before drawing a card wall to visualize workflow, it would be a good idea to sketch it or model it. Once the workflow is properly understood by sketching or modeling it, defining a card wall can be started by drawing columns on the board that represent the activities performed, in the order they are performed. During the first few weeks the team may make changes to the board until it stabilizes to fit their needs and criteria. It is also necessary for the activity steps to model both the in-progress and completed work, by convention this is done by splitting the column. Then the team can add the input queue and any downstream delivery steps that they wish to visualize. Also, they need to add buffers or queues that believe are necessary as demonstrated in figure 18.

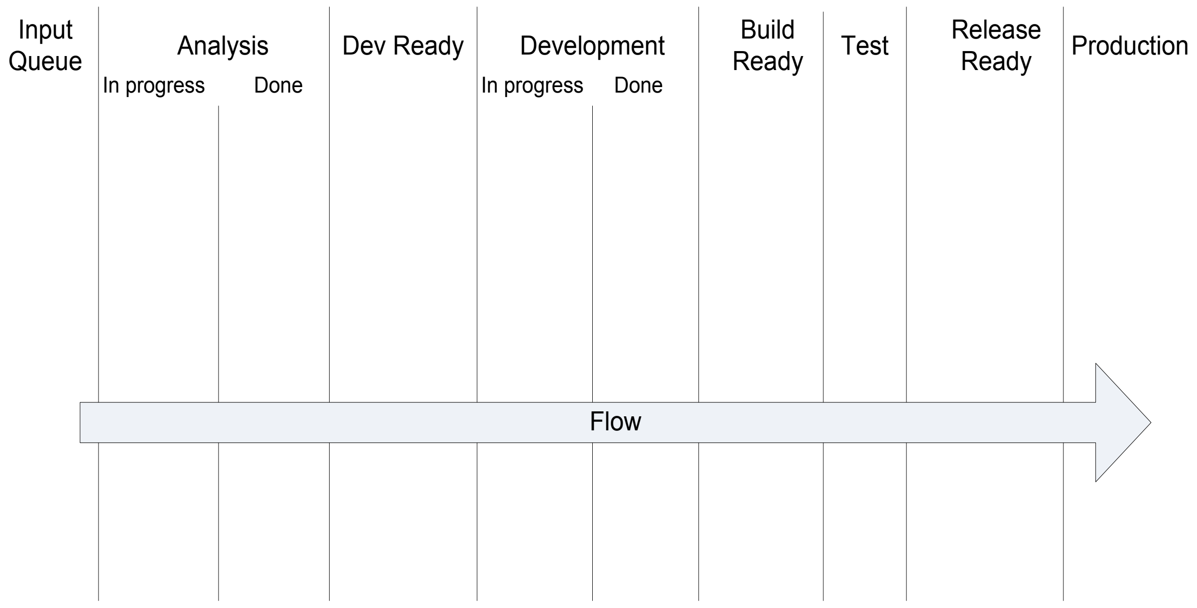

Figure 18: Kanban work flow (Anderson 2010)

4. The next step is to demand analysis. For each type of work identified, the team for the project under study should make a study of the demand based on historical data to make a quantitative study. Then the Kanban system can be designed and resourced appropriately to cope with this demand.
5. Once the team have an understanding of the demand, they can decide how to allocate capacity within the Kanban system to cope with that demand.
6. Each visual card representing a discrete piece of customer-valued work has several pieces of information on it. The design of the card is important. The information on the cards must facilitate the pull system and empower individuals to make their own pull decisions. For example, the card can have an electronic tracking number used to uniquely identify the item and to link it to the electronic version of the tracking system, the title of the item and the date the ticket entered the system. The date will serve a double purpose, it facilitate first-in, first-out queuing and it allows the team members to see how many days the card has been flowing. Also, the card can have the required delivery date. Some other information can be shown off-ticket, such as the name of the assigned person. As a general rule, the design of the ticket used to represent an individual piece of work that should have sufficient information to facilitate project-management decisions, such as the item to pull next, without the intervention or direction of a manager. The idea is to empower the team members with transparency of process, project goals and objectives, and risk information. Equally, Kanban, by empowering team members to make their own scheduling and prioritization decisions, shows respect for individuals and a trust in the system. A well designed work item card is a key enabler of a high-trust culture and a lean organization.

7. The team needs to align the design of the Kanban system and card wall with the decision made earlier to limit the boundary of work-in-progress control. Moreover, one common occurrence when designing a card wall for a Kanban system is a process in which two or more activities can happen concurrently. According to Anderson (2010), there are two basic patterns for coping with this situation. One is not to model it at all, just leave a single column where both activities can occur together. The other option is to split the board vertically into two sections. This situation can occur for the project under study when development ad test work in parallel as it will be discussed in the next improvement suggestion.
8. Another situation that Anderson (2010) mentioned and the team need to take into consideration is that there may be several activities that need to happen with a piece of customer-valued work, but those activities do not need to happen in any particular order. In these circumstances, it is important to realize that Kanban should not force the team to complete the activities in a given order. It is important that the Kanban system must reflect the way the real work is done. There are couple of strategies to the multiple unordered-activities problem. The first is similar to coping with concurrency: simply have a single column as a bucket for the activities and do not explicitly track on the board which of them is complete. The second, and potentially more powerful choice, is to model the activities in a similar fashion to the concurrent activities.
9. The team should also have a proper coordination with Kanban system.

6.4 How Can 'RK' Company Coordinate with Kanban Systems

Anderson (2010) identified several forms of coordination with Kanban system.

- The most popular form of coordination is the visual control and pull system. The work-in-progress limits are drawn on the board at the top of each column. Pull is signaled if the number of cards in a column is less than the indicated limit. When the team decides to pull an item, they can choose which item to pull based on available information. If something is blocked, the team can attach a pink ticket to the blocked item as an indication. The goal is to visually communicate enough information to make the system self-organizing and self-expediting at the team level. As a visual control mechanism, the Kanban board should enable team members to pull work without direction from their manager.
- Daily stand up meetings is another form of coordination. These meetings are a common element of agile process as discussed previously. However, the team can evolve the meetings and focus on the flow of work instead of who is working on what as it should be self-explanatory by the card wall. The facilitator will "walk the board" from right to left (in the direction of pull) through the tickets on the board. The facilitator might solicit a status update on a ticket or simply ask if there is any additional information that is not in the board and may not be known to the team. Particular emphasis will be placed on items that are blocked. Also about items that appear to be stuck and have not moved for a few days.
- Another form of coordination is the after meeting that consists of huddles of small groups of 2 or 3 people. This emerged as spontaneous behavior because team members wanted to discuss something on their minds: perhaps a blocking issue, a technical design or architecture issue or a process related issue. After meetings generate improvement ideas and result in process tailoring and innovation.

- Queue replenishment meetings serve the purpose of prioritization in Kanban. Queue replenishment meetings are held with product owners to fill the Kanban system's input queue for a single value stream.
- Release planning meetings happen specifically to plan downstream delivery. The person responsible for coordinating the delivery, usually a project manager, typically leads release planning meetings. Specialists are present for their technical knowledge and risk-assessment capabilities and managers are present so that decisions can be made. The outcome should be a completed template representing a release plan.
- Triage is used to classify bugs that will be fixed and their priority, versus bugs that will not be fixed and will be allowed to escape into production when the product is released. A typical defect triage involves a test lead, a test supervisor or manager, a development lead, a development supervisor or manager and a product owner. With Kanban it still makes sense to triage defects. However, the most useful application of triage is to the backlog of items waiting to enter the system. The purpose of a backlog triage is to go through each item on the backlog and decide whether it should remain in the backlog or be deleted. The reason for that is to reduce its size to facilitate easier prioritization discussions.
- When work items in the Kanban system is impeded, they will be marked as such and an issue work item will be created. The issue will remain open until the impediment is removed and the original work item can progress through the system. Reviewing open issues, therefore, becomes vital to improve flow through the system. While issues that are not progressing and are in themselves blocked or stale should be escalated to more senior management.

However, changing and optimizing the process is not sufficient for the project under study to achieve the required level of quality. The project has a lot of legacy features and complicated functionalities that makes it impossible to cover in regression with the current method of testing. Not to forget that the software system will continue to grow in advancements and complexity as new features and enhancements are presented with each iteration. This introduces many challenges on the quality assurance system test team.

Verifying the added features are functioning as required, ensuring that those changes didn't break any of the previous functionalities and validating bug fixes is nearly impossible to test manually. This is the current case of the project under study that causes the test team to change their goals on test pass or coverage in order to meet schedule making it difficult to reach acceptable levels of quality in the end.

Watts Humphrey stated that, "If you want to get a high quality product out of test, you have to put a high quality product into test".

'RK' Company can meet its intended level of quality by automating test cases where test execution time is much faster than manual test run, thus leading to maximizing test coverage.

When 'RK' company moves from waterfall to agile methodology with the regular change and ever evolving features, automation testing becomes a necessity as agile delivery without automation is not possible. Automation testing is the only way to ensure a very good coverage of both legacy features and new functionalities, performance and security testing.

6.5 How can ‘RK’ Company Start Automating

With the stated benefits that test automation provides, its principles, and its necessity in agile development, the automation team for the project under study needs to have a good understanding and a good start in order for the company to have bugs escaping to production close to zero and for the company to have a good return on investment.

It is crucial for the automation test team to have a very good understanding of the product they need to automate test cases for. They need to have a very good knowledge of the functionalities and features of the product and the technologies used to implement it.

Based on this knowledge, several decisions can be made:

1. The first decision to be made is to determine what test cases to automate and those that need to be tested manually. It is impossible to automate every possible scenario.

   Some types of bugs can be found only while someone is carefully watching the screen and running the application. These are the types of bugs that humans are vastly better at detecting than computers are (Page, Johnston, Rollison, 2009). Also test cases that are performed once or very few times might not be worth the cost and effort for automating. However, test cases that will be repeated many times or subject to human error are very good candidates for automation. Test cases that need to be run with different configurations or on different platforms can be executed faster if automated. Other possible candidates to consider for automation are the cases that require a lot of effort and time when done manually, or are impossible for manual testing. Moreover, functionality which is critical to the business can be automated as well. Nonfunctional tests such as load testing, stress testing and performance are good automation testing candidates.

2. Establishing the test automation criteria at the beginning will help the team to make consistent and better decisions about automation. The automation team for the project under study needs to carefully plan and design work by starting out to create an automation plan by defining their goal and by identifying the type of test cases to be automated.

3. To benefit from automation as much as possible, it is better for the automation team to be involved as early as possible in the software development life cycle and run the test more often. Being involved from the beginning allows the automation team to find bugs as early as possible, thus reducing the cost of fixing bugs radically.

   However, since the project under study has a lot of legacy features, it is impossible to automate all those features and still catch up with the development team as they will be adding new features at the same time. In this case, the automation team need to consider automating new features in order to catch those bugs for the new functionalities as early as possible and focus on automating core functionalities of legacy features. Also, they need to focus on the areas where most production bugs are found. Based on the results of the measure phase, the automation team will need to concentrate on the critical cases of the backend and functional areas while automating new features at the same time.

4. The next step for the automation team is to select the suitable tool or tools to use in automating test cases. Selecting the right tool is a crucial decision in automation. There are many tools in the market and it is important to choose the one(s) that best fit the project requirements.

   The automation team for the project under study needs to consider several key points that will help to make the right decision.

- The tool should support the platforms and technology that is used for the project implementation.
- The team might also consider to use the tool that uses the same programming language that the development team uses.
- The test tool should be stable enough as the automation team need to avoid false positive results as much as possible. This is the case when the test fails even though the targeted functionality is working correctly. Unstable tool can be one of the reasons for those false positives and as the number of test cases increase, the test team will not have sufficient time to investigate those failures.
- The richness of the tool features and at the same time the ease of use of the tool is another key point that the automation team needs to consider as it will affect the effort and time needed to learn how to use it.
- The tool should also have most of the features if not all that supports the verification of the functionalities for the project under test.
- Another key point to consider is the flexibility of the tool to be able to reuse, maintain and centralize the test code as much as possible as well as the ability of the tool to support any change in the user interface in order to avoid another type of false positive of the test result.

5. Another consideration is that the automation team lead should know the level of experience and skills for each team member in order to properly distribute the automation testing effort across the team members as the complexity of the test cases will have different levels.

6.6 How Can the Automation Team Create Good Quality Test

After the automation team for the project under test identifies the goal and test strategy as well as selecting the right tool for testing the project, it is important to have a good test design and quality.

- It is important to have each automated test case with a single objective and not have the same test verifying several expectations as bugs tend to hide each other. Once the first bug is fixed, the automated test will need to be run again and reveal the second bug, while if each test is designed to verify a single expectation, multiple bugs can be found by running several tests at the same time.
- Also, it is better to keep the test small. Large and complex automated tests are difficult to maintain and debug. Also, having a small test case will allow the team to share reusable test code and test data.
- Another key point to consider is to keep test cases independent from each other in order to avoid unnecessary test failures. If one of the test cases fail, the rest of the dependent test cases will fail as a consequence and will be blocked from verifying other functionalities until that specific failure is fixed.
- It is also important to group and organize the test in a specific logic to be able to identify easily. As the number of automated test increases, it becomes difficult to find a specific test. For example, test cases can be organized by the application features.
- Automation test team need to avoid redundant test cases. Test cases with different input values that will be validating the same code path should not be added. For example, in the case of testing a field that takes a specific range of numbers, it is better to verify the boundary values of that field and not to test every single possible input.

- Automation test team need to centralize reusable test code as much as possible. If a single functionality changes between builds, many test cases will be affected and will be broken. It will be more efficient and less time consuming to fix the change in one place and not in each and every single affected test case.
- It is important to avoid fragile user interface automation test tools that depend on the location or coordinates to locate an element in the application. The reason is that, if the developer decided to change the location of a specific control, the automated test cases will no longer be able to find the object and as a consequence will fail. Other tools that are able to identify the control by its Id (which is a unique value) are more stable. The Id of the element is rare to change which will make the automated tests more resistant to user interface changes and thus reduces the maintenance required for the test.
- Automation test team needs to reduce test code maintenance as much as possible in order to invest time in adding test cases and increase code coverage rather than spend the time to fix broken automated test cases.
- Other key point that the automation team of the project under study needs to consider, is to automate the test cases in parallel as the development team is implementing the specific feature. The automated test cases can be executed after implementation is done. This will help the team to reduce the lead time and reduce the release cycle time.
- It is also important to enhance the test execution time. As the number of test cases increase with time, the execution time will increase as well. In this case the test engineers need to improve the performance by looking into parallelizing the test run, which means that they will need to run multiple test cases concurrently. This will reduce the test time significantly, test the application more efficiently and will be able to execute those tests more frequently.

By designing the test cases to be independent from each other will be the key point for having those tests run in parallel and not sequentially.

Another improvement suggestion that 'RK' company can apply or any software organization, is to improve the software quality and CTQ performance at design phase. This can be done by using a quality tool "Quality Function Deployment" (QFD) that can be considered as a preventive action that results in a reduction in the cost.

6.7 How can 'RK' Company Implement Quality Function Deployment

QFD is focused on preventive actions. It prevents or minimizes the causes of design problems or defects instead of reacting to them at a later stage in the software development cycle. This leads to a radically reduced cost, reduced cycle time and improved customer experience as early design changes for a software feature are much less expensive to make than after the release to production (Bernal et al., 2009).

A set of matrixes is used to relate the voice of the customer to the product's technical requirements and component requirements. This matrix is called the "House of Quality".

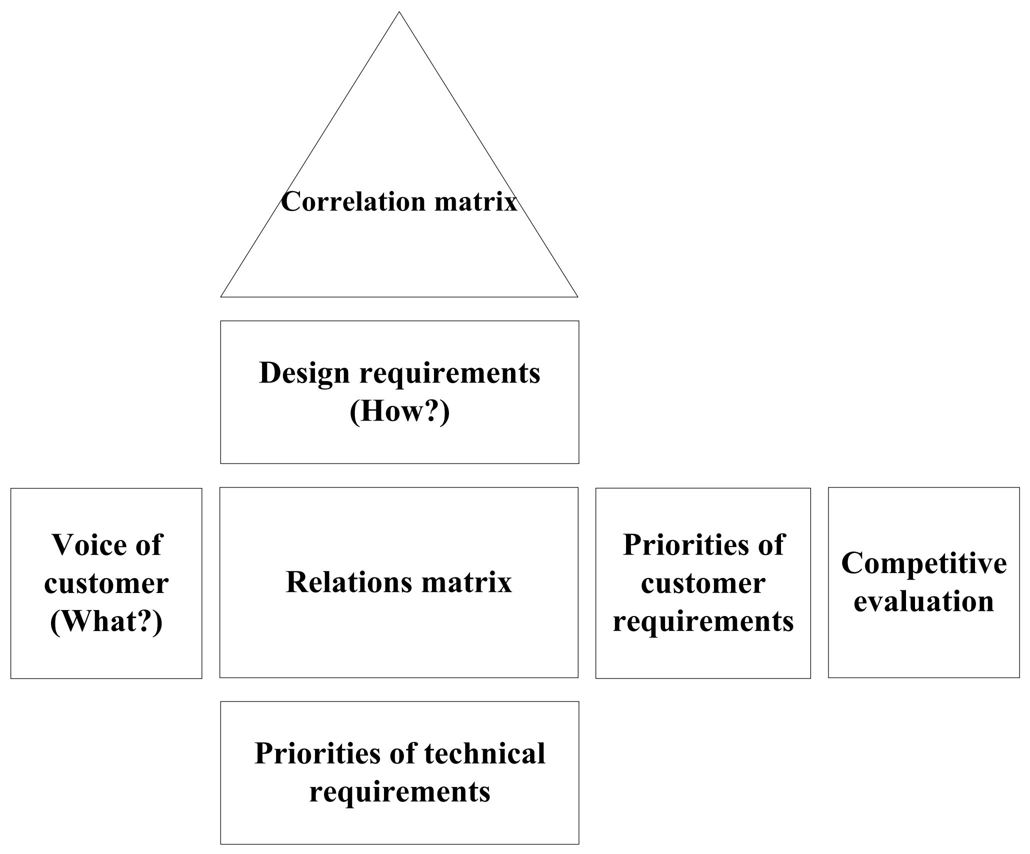
Correlation matrix
Design requirements (How?)
Voice of customer (What?)
Relations matrix
Priorities of customer requirements
Competitive evaluation
Priorities of technical requirements

Figure 19: Quality Function Deployment

As shown in the figure 19, 'RK' company needs to follow the below steps to build the house of quality:

1. Identify customer requirements

QFD starts with establishment of objectives, which represent the answer to "What?" what is desired in order to reach the new service's development? These objectives derive from client's requirements and are called the "voice of the customer" (Bernal et al., 2009).

Sometimes the client requirements are general, vague and difficult to implement directly, a more detailed description is needed. These are three kind of service characteristics that must be differentiated. The requirements mentioned directly by the clients will be called "performance requirements"; other wants are difficult for clients to verbalize. These "wants" are essential parts of the service and perform basic functions that the user expects and consider as given. These basic functions are known as “basic requirements”. The third kind of service feature is an “emotional requirement”, it reflects a need that the client has not appreciated before.

2. Identify technical requirements

Technical requirements are design characteristics that describe the customer requirements as expressed in the language of the designer or engineer. Essentially, they are the “How” by which the company will respond to the “What” – customer requirements. These are measurable features that can be evaluated at the end of development process (Evans & Lindsay, 2005).

3. Develop a relationship matrix

A relationship matrix should be developed between customer requirements and the technical requirements.

Relations between the client and design requirements are not always 1:1, there are complex relationships and varying levels of strength. A single design requirement may have an influence on several of the client’s requirements (Bernal et al., 2009).

However, the lack of a strong relationship between a customer requirement and any technical requirement shows that the customer needs are either not addressed or that the final design will have difficulty in meeting them. Similarly, if a technical requirement does not affect any customer requirement, it may be redundant or the designers may have missed some important customer need.

4. Add key competitor evaluation and key selling points

This step identifies importance ratings for each customer requirement and evaluates competitor’s existing products or services for each of them. Customer importance ratings represent the areas of greatest interest and highest expectations as expressed by the customer. Competitive evaluation highlights the absolute strengths and weaknesses in competing products. By using this step, designers can discover opportunities for improvement. It also links QFD to a company's strategic vision and indicates priorities for the design process

5. Evaluate technical requirements of competitive products and services and develop targets

This step is usually accomplished through intelligence gathering or product testing and then translated into measureable terms. These evaluations are compared with the competitive evaluation of customer requirements and technical requirements. If a competing product is found to best satisfy a customer requirement but the evaluation of the related technical requirements indicates otherwise, then either the measures used are faulty or else the product has an image difference, which affects customer perceptions. On the basis of customer importance ratings and existing product strengths and weaknesses, targets for each technical requirement are set.

6. Selecting requirements to be deployed in the remainder of the process

The technical requirements that have a strong relationship to customer needs, have poor competitive performance, or are strong selling points are identified during this step. These characteristics have the highest priority and need to be "deployed" throughout the remainder of the design and development process to maintain a responsiveness to the voice customer. Those characteristics not identified as critical do not need such rigorous attention (Evans & Lindsay, 2005).

7 Control Phase

'RK' Company main goal to ensure the success of the project under study is to improve its quality by reducing the number of bugs found in production to nearly zero. Several improvement suggestions were provided in the improvement phase.

The next step for 'RK' company is to start implementing those changes and measure the improvement performance in order to take the necessary actions that will ensure that the process is under control.

Several metrics and reports were explained in details in the literature review chapter.

- Tracking the Work-in-progress

By measuring the number of work items in progress at each stage in the system after every release, can help to take the corrective actions where needed. The expected result is to see the system flowing smoothly and the height of the bands on the chart are stable.

- Lead Time

Another identified metric is the lead time. By measuring the average time of how long items take until they reach production from the time they get approved, can help measure how predictably 'RK' company delivers. It is expected that the average lead time is similar in each cycle.

- Throughput

Throughput is another mean of measurement and it helps to indicate the number of items that got delivered in a given period of time.

- Number of bugs

In addition, measuring the number of bugs that escaped to production helps identify if 'RK' company is meeting the identified goal. This number is expected to fall to close to zero.

‘RK’ Company should not only focus on process measurements but it should also measure testing quality. For this, ‘RK’ company can use the defect removal efficiency that was also explained in the literature review chapter.

- Defect removal efficiency

  Ideally ‘RK’ company wants the defect removal efficiency to be 1, which means that there are no defects found after delivery. However, achieving zero defects after delivery is nearly impossible. For this, it is necessary for the test team to find as many bugs as possible before delivery and not to drop the measure of defect removal efficiency below 0.95.

It is important for ‘RK’ company to monitor the applied improvements, use the suggested measurements and metrics and take the necessary actions when needed to ensure that it reaches the specified goals that will lead to the success of the project under study.

CHAPTER 5:

CONCLUSIONS AND FUTURE WORKS

1 SWOT Analysis

In this thesis we addressed the problem of software quality management and proposed a DMAIC based approach. The SWOT analysis of the proposed approach is as follows:

Strengths

This study demonstrated the importance of six sigma as a business strategy that focus on eliminating inefficiency through the use of a systematic approach. It also revealed its ability to pursue to find and eliminate the causes of errors. This allows organizations to identify and implement improvements that leads to an increased confidence in the quality of the product produced at all levels: team, management, marketing and most importantly the customer especially in the current competitive market that demands short cycle time, fast software releases with feature rich and high quality software products.

Weaknesses

Despite the defined strengths, six sigma has its weaknesses as well. It requires the total cooperation of the organization at all levels. It also relies on good data for understanding process performance, thus, it requires a considerable effort to be made to collect accurate data which makes it time consuming and costly.

Opportunities

One of the key reasons to pursue six sigma DMAIC methodology is to be a head and distinct in the competitive market. It helps to put a great emphasis on speed, quality and productivity. Also, it provides a customer-driven excellence by focusing on the customer requirements through high quality software products and as a result, it improves the organizational return on investment. Moreover, it improves the overall performance of an organization by providing a working methodology that allows the organization to fulfill its plan in order to achieve its goals.

Threats

There are threats that can weaken the success of six sigma DMAIC methodology such as a dysfunctional organizational culture whose shared values and behavior are at odds with its long term health. Also, lack of creativity and cooperation of the workforce can also be considered as a threat as the workforce is a principal source of innovative ideas that is a necessity in the improvement phase of DMAIC.

Strengths: 1. Focus on eliminating inefficiency 2. Assist in identifying the root causes of defects 3. Assist in executing quality improvement efforts	Weaknesses: 1. Requires total cooperation of the company 2. Requires significant amount of data collection and analysis
Opportunities: 1. Distinction in competitive market 2. Provides customer-Driven excellence 3. Improves return on investment 4. Improves overall performance of the organization	Threats: 1. Dysfunctional organizational culture 2. Lack of creativity and cooperation of the workforce.

Table 17: SWOT analysis of DMAIC

2 Future Works

In the software development industry, there is always need for improvement. Recent studies show that Agile and Lean strategies are more effective than traditional strategies on average. However, the success rate for software projects is still low compared to other industries. There is still work needed on finding ways to reduce cycle times especially when implementing large features that would increase the time needed for development and testing, thus leading to a reduced predictability for project release dates. Also, software companies still face problems to find the proper tools and methods that would help facilitate the design phase especially for features with complicated implementation logic and the necessity to keep the software product easy to use at the same time. This causes the development team to go back to design phase in the middle of implementation and as a consequence, it leads to increased cycle time and reduced predictability. Moreover, Agile and lean strategies, require team members to be experienced enough to be able to make decisions with the least cost of failure, which is not the case in most software teams where team members expertise vary. Another possible area that requires improvement is the ability to find reliable and easy to use automation test tools that will not cause false positive test results that would require a lot of investigation and maintenance from the test automation team that will also lead to an increased cycle time.

REFERENCES

[1] Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods Review and Analysis. Retrieved February 17, 2015, from: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

[2] Alexander, M. (2013). When Little Objective Data are Available, Find Root Causes and Effects with Interrelationship Digraphs and JMP. Proceedings of the SouthEast SAS Users Group (SESUG) Conference.

[3] Anderson, D. J. (2004). Agile Management for Software Engineering. Prentice Hall, Upper Saddle River, New Jersey.

[4] Anderson, D. (2010). Kanban Successful Evolutionary Change for Your Technology Business. Blue Hole Press, Sequim, Washington.

[5] Bach, J. Agile Test Automation. Retrieved February 17, 2015, from: http://www.satisfice.com/articles/agileauto-paper.pdf

[6] Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. International Journal of Engineering & Technology 2(5), 742-749.

[7] Bernal, L., Dornberger, U., Suvelza, A., & Byrnes, T. (2009). Quality Function Deployment (QFD) For Services. Retrieved February 17, 2015, from: http://www.vgu.edu.vn/fileadmin/pictures/studies/MBA/Handbook_QFD_Services.pdf

[8] Blakeslee, Jr. J. A. (1999). Implementing the Six Sigma solution. Quality Progress 32 (7), 77-85.

[9] Beck, K. (1999). Embracing Change with Extreme Programming. IEEE Computer 32(10), 70-77.

[10] Boehm, B., & Turner, R. (2005). Management Challenges to Implement Agile Processes in Traditional Development Organization. IEEE Software 22(5), 30-39.

[11] Boehm, B. (2002). Get Ready for The Agile Methods, With Care. Computer 35(1), 64-69.

[12] Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A Framework for Adapting Agile Development Methodologies. European Journal of Information Systems 18(4), 332-343.

[13] Chow, T., & Cao, D. B (2008). A Survey Study of Critical Success Factors in Agile Software Projects. The Journal of Systems and Software 81(6), 961-971.

[14] Chang, C. W., Wu, C. R., & Lin, H. L. (2006). A Simplified Measurement Scheme for Software Quality. Journal of Information and Optimization Science 27(3), 723-732.

[15] Chakrabarty, A., & Chuan Tan, K. (2007). The Current State of Six Sigma Application on Services. Managing Service Quality 17(2), 194-208.

[16] Chan, F.K.Y., & Thong, J.Y.L. (2009). Acceptance of Agile Methodologies: A Critical Review and Conceptual Framework. Decision Support Systems 46(4), 803-814.

[17] Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. Information Systems Research 20 (3), 329-354.

[18] Dawson, S. P. (1994). Continuous Improvement in Action Applying Quality Principles to Software. Information Systems Management 11(1), 31-39.

[19] Davison, A. L., & Al-shaghana, K. (2007).The Link between Six Sigma and Quality Culture an Empirical Study. Total Quality Management 18(3), 249-265.

[20] Daneva, M., Veen, E. V. D., Amrit, C., Ghaisas, S., Sikkel, K., Kumar, R., Ajmeri, N., Ramteerthkar, U., & Wieringa, R. (2013). Agile Requirements Prioritization in Large-Scale Outsourced System Projects: An Empirical Study. The Journal of Systems and Software 86(5), 1333-1353.

[21] Desai, T. N., & Shrivastava, R. L. (2008). Six Sigma – A New Direction to Quality and Productivity Management, Proceedings of the World Congress on Engineering and Computer Science 2008 WCECS 2008, October 22 - 24, 2008, San Francisco, USA.

[22] Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A Decade of Agile Methodologies: Towards Explaining Agile Software Development. Journal of Systems and Software 85(6), 1213-1221.

[23] Dustin, E., Garett, T., & Gauf, B. (2009). Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley, Boston.

[24] Drury, M., Conboy, K., & Power, K. (2012). Obstacles to Decision Making in Agile Software Development Teams. The Journal of Systems and Software 85(6), 1239-1254.

[25] Dyba, T., & Dingsoyr, T. (2008). Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology 50(9-10), 833-859.

[26] Evans, J. R., & Lindsay, W. M. (2005). An Introduction to Six Sigma & Process Improvement. Thomson South-Western, Mason, OH.

[27] Evans A., J.P., & Lindsay, W.M. (2008). Managing for Quality and Performance Excellence, seventh ed. Thomson South-Western, Mason, OH.

[28] Edison, H., Bin Ali, N., & Torkar, R. (2013). Towards Innovation Measurement in the Software Industry. The Journal of Systems and Software 86(5), 1390-1407.

[29] Fewster, M., & Dorothy, G. (1999). Software Test Automation, Effective use of test execution tools. Addison-Wesley, Boston.

[30] Fontana, R. N., Fontana, I. M., Garbuio, P. A. D. R., Reinehr, S., & Malucelli, A. (2014). Processes versus People: How Should Agile Software Development Maturity be Defined? The Journal of Systems and Software 97, 140-155.

[31] Galin, D. (2004). Software Quality Assurance from Theory to Implementation. Pearson Addison-Wesley, New York.

[32] Garvin, D. (1984). What Does Product Quality Really Mean? Sloan Management Review, 25-43.

[33] Goh, T., & Xie, M. (2004). Improving on the Six Sigma Paradigm. The TQM Magazine 16(4), 235-240.

[34] Glass, R. L. (2001). Agile Versus Traditional: Make Love, Not War! Cutter IT Journal 14(12), 12-18.

[35] Gentleman, R., Hornik, K., & Parmigiani, G. (2012). Six Sigma with R Statistical Engineering for Process Improvement. Springer, New York.

[36] Grenning, J. (2001). Launching XP at a Process-Intensive Company. IEEE Software 18(6), 3-9.

[37] Highsmith, J. (2002). Agile Software Development ecosystems. Pearson Edition, Boston, MA.

[38] Hossain, A., Kashem, M., & Sultana, S. (2013). Enhancing Software Quality Using Agile Techniques. IOSR Journal of Computer Engineering 10(2), 87-93.

[39] Hooda, R. V. (2012). An Automation of Software Testing: A Foundation for the Future. International Journal of Latest Research in Science and Technology 1(2), 152-154.

[40] Hung, H. C., & Sung, M. H. (2011). Applying Six Sigma to Manufacturing Process in the Food Industry to Reduce Quality Cost. Scientific Research and Essays 6(3), 580-591.

[41] Ilie, G., & Ciocoiu, C. N. (2010). Application of Fishbone Diagram to Determine the Risk of an Event with Multiple Causes. Management Research and Practice 2(1), 1-20.

[42] Issac, G., Rajendran, C., & Anantharaman, R. N. (2010). Determinants of Software Quality: Customer's Perspective. Total Quality Management & Business Excellence 14(9), 1053-1070.

[43] Issac, G., Rajendran, C., & Anantharaman, R. N. (2004). A Conceptual Framework for Total Quality Management in Software Organizations. Total Quality Management & Business Excellence 15(3), 307-344.

[44] Jayanthi, R., & Florence, M. L. (2013). A study on Software Metrics and Phase Based Defect Removal Pattern Technique for Project Management. International Journal of Soft Computing and Engineering 3(4), 151-155.

[45] Johannsen, F., Leist, S., & Zellner, G. (2011). Six Sigma as a Business Process Management Method in Services: Analysis of the Key Application Problems. Information Systems and e-Business Management, 9(3), 307-332.

[46] Jones, C. (2008). Measuring Defect Potentials and Defect Removal Efficiency. The Journal of Defence Software Engineering 21(6), 11-13.

[47] Kabir, E., Boby, M. I., & Lutfi, M. (2013). Productivity Improvement by Using Six Sigma. Engineering and Technology 3(12), 1056-1084.

[48] Koripadu, M., & Subbaiah, K. V. (2014). Problem Solving Management Using Six Sigma Tools & Techniques. International Journal of Scientific and Technology Research 3(2), 91-93.

[49] Korkala, M., & Maurer, F. (2014). Waste Identification as the Means for Improving Communication in Globally Distributed Agile Software Development. The Journal of Systems and Software 9, 122-140.

[50] Kwak, Y. H., & Anbari, F. T. (2006). Benefits, Obstacles, and Future of Six Sigma Approach. Technovation, 26(5-6), 708-715.

[51] Larman, C. (2004). Agile & Iterative Development: A Manager’s Guide. Addison-Wesley, Boston.

[52] Lehtinen, T. O. A., Mantyla, M. V., Vanhanen, J., Itkonen, J., & Lassenius, C. (2014). Perceived Causes of Software Project Failures – An Analysis of Their Relationships. Information and Software Technology 56(6), 623-643.

[53] Lehtinen, T. O. A., Mantyla, M. V., & Vanhanen, J. (2011). Development and Evaluation of a Lightweight Root Cause Analysis Method (ARCA method) – Field Studies at Four Software Companies. Information and Software Technology 53(10), 1045-1061.

[54] Lyu, J. J., & Liang, C. C. (2014). Effective Approach to Quality Control for Small-Medium Software Companies. Total Quality Management & Business Excellence 25(3-4), 296-308.

[55] Lyytinen, K., & Rose, G.M. (2006). Information System Development Agility as Organizational Learning. European Journal of Information Systems 15(2), 183-199.

[56] Marek, R. P., Elkins, D. A., & Smith, D. R. (2001). Understanding the Fundamentals of Kanban and Conwip Pull System Using Simulations. In B. A. Peters, J. S. Smith, D. J. Medeiros, & M. W. Rohrer, (Eds.) Proceedings of the 2001 Winter Simulation Conference (pp. 921-929). Texas, USA: A&M University.

[57] Maheshwari, S., & Jain, D. C. (2012). A Comparative Analysis of Different Types of Models in Software Development Life Cycle. International Journal of Advanced Research in Computer Science and Software Engineering 2(5), 285-290.

[58] Mast, J. D., & Lokkerbol, J. (2012). An Analysis of the Six Sigma DMAIC Method from the Perspective of Problem Solving. International Journal of Production Economics, 139(2), 134-154.

[59] Marques, P. A., & Requeijo, J. G. (2009). SIPOC: A Six Sigma Tool Helping on ISO 9000 Quality Management Systems. 3rd International Conference on Industrial Engineering and Industrial Management (pp. 1229-1238).

[60] Mafakheri, F., Nasiri, F., & Mousavi, M. (2008). Project Agility Assessment: An Integrated Decision Analysis Approach. Production Planning & Control 19(6), 567-576.

[61] Meso, P., & Jain, R. (2006). Agile Software Development: Adaptive Systems Principles and Best Practices. Information Systems Management 23(3), 19-30.

[62] Middleton, P. (2001). Lean Software Development: Two Case Studies. Software Quality Journal 9(4), 241-252.

[63] Moe, N. B., Aurum, A., & Dybå, T. (2012). Challenges of Shared Decision-Making: A multiple Case Study of Agile Software Development. Information and Software Technology 54(8), 853–865.

[64] Moe, N. B., Dingsøyr, T., & Dybå, T. (2009). Overcoming Barriers to Self-Management in Software teams. IEEE Software 26 (6), 20–26.

[65] Moe, N. B., Dingsøyr, T., & Dybå, T. (2010). A Teamwork Model for Understanding an Agile Team: A Case Study of a Scrum Project. Information and Software Technology 52(5), 480-491.

[66] Munassar, N. M. A., & Govardhan, A. (2010). A Comparison between Five Models of Software Engineering. International Journal of Computer Science 7(5), 94-101.

[67] Miller, D., & Lee, J. (2001). The People Make the Process: Commitment to Employees, Decision Making, and Performance. Journal of Management 27, 163-189.

[68] Misra, S. C., Kumar, V., & Kumar, U. (2009). Identifying Some Important Success Factors in Adopting Agile Software Development Practices. The Journal of Systems and Software 82(11), 1869-1890.

[69] Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. Communications of the ACM 48 (5), 73-78.

[70] O'Neill, D. M., & Duvall, C. (2003). A Six Sigma Quality Approach to Workplace Evaluation. Journal of Facilities Management, 3(3), 240-253.

[71] Pressman, R. S. (2001), Software Engineering: A Practitioner's Approach, 5th edition, McGraw Hill.

[72] Page, A., Johnston, K., & Rollison B. (2009). How We Test Software at Microsoft. Microsoft Press, California.

[73] Pai, W. C. (2002). A Quality-Enhancing Software Function Deployment Model. Information Systems Management 19(3), 20-24.

[74] Phan, D. D. (2001). Software Quality and Management: How the World's Most Powerful Software Makers Do It. Information Systems Management 18(1), 56-67.

[75] Quyoum, A., Din Dar, M. U., & Quadri, S. M. K. (2010). Improving Software Reliability Using Software Engineering Approach – A Review. International Journal of Computer Applications 10(5), 41-47.

[76] Reel, J. S. (1999). Critical Success Factors in Software Projects. IEEE Software 16(3), 18-23.

[77] Rising, L., & Janoff, N. S. (2000). The Scrum Software Development Process for Small Teams. IEEE Software 17(4), 26-32.

[78] Reiffer, D.J. (2002). How Good are Agile Methods? IEEE Software 19(4), 14–17.

[79] Rainer, A., & Hall, T. (2002). Key Success Factors for Implementing Software Process Improvement: A Maturity-Based Analysis. Journal of Systems and Software 62 (2), 71–84.

[80] Schwaber, K. (1995). Scrum Development Process. OOPSLA 95 workshop on Business Object Design and Implementation. Verlag.

[81] Scarpa, M., & Puliafito, A. (2009). Developing High Quality Software. International Journal of Parallel, Emergent and Distributed Systems 24(2), 171-187.

[82] Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile Processes and Methodologies: A conceptual Study. International Journal on Computer Science and Engineering 4(5), 892-898.

[83] Stankovic, D., Nikolic, V., Djordjevic, M., & Cao, D.B. (2013). A Survey Study of Critical Success Factors in Agile Software Projects in Former Yugoslavia IT Companies. The Journal of Systems and Software 86(6), 1663-1678.

[84] Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in Co-Located Agile Software Development Projects. The Journal of Systems and Software 85(6), 1222-1238.

[85] Snee, R. (2004). Six Sigma: the Evolution of 100 Years of Business Improvement Methodology. International Journal of Six Sigma and Competitive Advantage 1(1), 4-20.

[86] Sridhar, V. (2003). Implementing Automated Testing. Idea Group Inc., Hershey, PA, USA.

[87] Subramanian, G. H., Jiang, J. J., & Klein, G. (2007). Software Quality and IS Project Performance Improvements from Software Development Process Maturity and IS Implementation Strategies. The Journal of Systems and Software 80(4), 616-627.

[88] Tarhan, A., & Yilmaz, S. G. (2014). Systematic Analyses and Comparison of Development Performance and Product Quality of Incremental Process and Agile Process. Information and Software Technology 56(5), 477-494.

[89] Thackeray, R., & Van Treeck, G. (2007). Applying Quality Function Deployment for Software Product Development. Journal of Engineering Design 1(4), 389-410.

[90] Wougang, I., Akinladejo, F. O., White, D. W., Obaidat, M. S., Fellow of IEEE, & Fellow of CS. Coding-Error Based Defects in Enterprise Resource Planning Software: Prevention, Discovery, Elimination and Mitigation. The Journal of Systems and Software 85(7), 1682-1698.

[91] Yu, X., & Petter, S. (2014). Understanding Agile Software Development Practices Using Shared Mental model Theory. Information and Software Technology 56(8), 911-921.

[92] Zimmerman, J.P., & Weiss, J. (2005). Six Sigma's Seven Deadly Sins. Quality 44, 62-66.

[93] Zu, X., Fredendall, L. D., & Douglas, T. J. (2008). The Evolving Theory of Quality Management: The Role of Six Sigma, Journal of Operations Management, 26(5), 630-650.

APPENDIX

The below is a set of questions that were asked to the Development and QA managers for the project under study at 'RK' company in order to get their feedback on the issues that their teams are facing.

1. What is the current methodology used for the project?
2. How does the testing process works? At what stage testing starts?
3. What is the percentage of manual test vs. automated test?
4. Has the project faced situations where major / critical bugs reached to production that should have been caught by testers?
5. How long regression takes?
6. How regression test cases are defined / selected?
7. What are the problems that cause the release date to be postponed?
8. Does the dev. Team create unit test?
9. Is there any available documentation that the test team rely on?
10. Do you believe that these documents are clear enough and complete?
11. How often these documents are updated / maintained?
12. How does the QA team communicate with the Dev team?
13. What kind of tests are performed?
14. Do you believe that management is supplying adequate test resources?
15. Do you believe that the test schedule is adequate for the amount of testing that should be done?
16. How much time is spent on test maintenance?

17. In your opinion what are the main issues that exist in the current Test / Development process?