Software Quality Assurance in Scrum Projects: a case study of development processes among scrum teams in South Africa

By

Andile Koka

Thesis Submitted in fulfilment of the requirements for the degree

MASTER OF TECHNOLOGY

In

Information Technology

In the

FACULTY OF INFORMATICS AND DESIGN

at the

CAPE PENINSULA UNIVERSITY OF TECHNONLOGY

Supervisor: Prof. Nhlanhla Mlitwa

August 2015

DECLARATION

I, Andile Isaac Koka (213304376) declare that “Software Quality Assurance in Scrum Projects: a case study of development processes among Scrum teams in South Africa” is my own work and it has not been previously submitted to another University or for another qualification.

26 August 2015

Signed **Date**

ABSTRACT

The use of Information and Communication Technology (ICT) in business has evolved to such an extent that many organizations (if not all) rely on Information Technology (IT) systems to better manage their processes, get competitive advantage, improve performance (efficiency and effectiveness), provide quality services on time and most importantly to keep customers happy. This has changed the way people communicate and conduct businesses, lowering processing cost, time and improving a return on investment. Therefore, high quality software systems are essential. Organizations adopt Agile Scrum methodologies in order to develop applications that help them to obtain a return on investment quickly, to improve customer satisfaction and to maintain competitive advantage. However, the IT industry is yet to develop error-free software that meets the expected quality standards. Therefore, the aim of this study was to explore the extent to which software quality assurance measures can be understood and applied to maximize the quality of software projects developed under Scrum methodology.

A qualitative research method informed by an interpretive approach was used to collect and analyse data. Following the purposive sampling technique, five Scrum teams operating in different environments and two academics from one academic institution were interviewed. Structuration Theory (ST) was then used as an analytical framework to analyse data and to improve the understanding of Scrum practices and related quality assurance (QA) processes. Drawing on the major terms of ST, the contextual terrain of the Scrum development process was mapped. It reflected that rules are important aspects of Scrum functions. However, rules are not as strictly applied as in the traditional methodologies. The developer skill, project type and size have a direct influence on the practice/s. In Scrum, rules are flexible in that they can be modified to meet the environment and conditions of the team. Equally significant are resources, most particularly, time and the human resources in the form of developers and Scrum leaders. Otherwise, unit testing, user acceptance testing, close collaboration and code reviews were perceived as the most important practices in Scrum projects.

In view of the findings, recommendations can be summed up into 4 main points; (1) that to ensure quality assurance in Scrum, Scrum teams, especially team leaders, should enforce compliance to standards, regardless of time pressures and tight deadlines; (2) It seems that the practice of working with the client to test final products as a quality assurance mechanism is working for all parties. This practice is encouraged and must be maintained; (3) Code reviews must be enforced, and that organisations invest in resources including the constant training of developers; (4) Project product owners, project managers, team leaders and business analysts should regularly meet with the user to verify requirements prior to the implementation phase. Active stakeholder involvement can minimize development costs and time.

Key words: Agile software development, Scrum methods, software quality assurance, structuration theory

ACKNOWLEDGEMENTS

I would like to thank the following people for their special contributions in my life:

- My supervisor Professor Mlitwa, thank you for your advice, sacrifices and the words of wisdom and encouragement.
- My mother Miss Nonzame Koka, and the rest of my family, thank you for your support and for believing in me.
- My Aunt Nomathamsanqa Tamie Koka, sisi without the sacrifices, support and faith I would not be where I am today, may God bless you.
- All the participants who sacrificed their valuable time giving input in my study, hope you will do the same to other researchers.
- My classmate, Ayanda Pekane, thank you for the constant encouragement and support throughout the journey of this thesis.

Lastly I would like to thank the Cape Peninsula University of technology for the financial support through this study.

LIST OF ABBREVIATIONS

CPUT	-	Cape Peninsula University of Technology
CAS	-	Complex Adaptive System
E-Learning	-	Electronic Learning
EOHMC	-	EOH Microsoft Coastal
GT	-	Grounded Theory
IS	-	Information Systems
ISD	-	Information Systems Development
IT	-	Information Technology
ITIL	-	Information Technology Infrastructure Library
NMMU	-	Nelson Mandela Metropolitan University
PC	-	Personal Computer
QA	-	Quality Assurance
SD	-	Software Development
SDLC	-	Software Development Life Cycle
SE	-	Software Engineering
SQA	-	Software Quality Assurance
ST	-	Structuration Theory
TDD	-	Test Driven Development
XP	-	Extreme Programming

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE – INTRODUCTION

1. Importance of Information Technology

Information Technology (IT) plays a significant role in our societies. In the past few years, the use of mobile devices such as cell phones, tablet PCs, and laptops has evolved in such a way that individuals cannot imagine life without them (Kent, 2013). Mobile devices have helped individuals to access information inexpensively, by means of such connections, users save on travelling costs (Jagun & Heeks, 2007). Mobile devices have made it possible for users to make calls, text, store information (personal and work related information), access the Internet and social networks and to use online financial transactions, easily and quickly.

From the business perspective, technology plays a significant role in that companies are now relying on computers and software to effectively manage business information, to the extent that almost all organizations incorporate IT solutions to operate successful (Bae & Ashcroft, 2004). The enormous increase in the use of IT in our societies therefore, emphasizes the significance of quality in IT projects. Software products, related development processes and the quality therefore, are central to this point thus the quality of software products and related development processes is central in these projects (Huo, Verner, Zue, & Barbar, 2004; Serena, 2007).

The IT industry is still faced with the challenge of delivering error-free software that meets the expected quality standards (Pressman, 2010). The development of software projects involves numerous error-prone activities. Failure to identify and correct such errors can result in subsequent failures in mission critical situations like healthcare systems, nuclear reactions, telephone switchboards, activities in space, banking and airline traffic control, thus the quality of software products is important (Huo et al., 2004; Serena Software Inc, 2007). Quality in software products is understood to be the most important determining factor in the success of IT projects (Pressman, 2010). Customers rate a project successful in terms of its ability to satisfy or exceed their needs, in other words the questions that are asked are: how well does the product serve the customers? Does it cover all the specified requirements? Is the project delivered on time and within budget? However, in other mission critical industries like the healthcare sector, success is strongly determined by the software's quality in terms of its efficiency and effectiveness. In this instance efficiency relates to the accuracy of the software in achieving its intended task. Effectiveness on the other hand refers to ensuring that the software accurately produces the product that satisfies the stated customer needs (Ichu & Nemani, 2011).

As a basis of almost all electronic task-based instructions therefore, software (either application or system software) is a significant component of computerized processes (Fetaji & Fetaji, 2009). The quality of software as a function of software engineering thus is implied in this argument. That is, if software is to accurately execute task functions, its development process should be prudent so as to yield as near bug-free code as is technically possible (Ullah & Zaidi, 2009; Post & Kendall 2004; Szalvay, 2004). Otherwise, without dexterity in the development process, the quality of a software product will be suspect with a resultant program possibly failing to work effectively (Mahanti, 2007). However, the developers' mere noble intentions alone are not enough to attain high quality software. The skill, experience and reputation of the developer are also critical (Khalaf & Al-Jedaiah, 2008). So is the management of the development process and most significantly, an appropriate software development methodology as well as effective software quality-assurance measures (Salo & Abrahamsson, 2006; Leau, Loo, Tham & Tan, 2012).

2. Extent of Quality Assurance in Agile Methods

It is stated in the preceding sections that software quality is growing in prominence, at a rate of unprecedented proportions, most probably due to the "pace of technology change and the underlying, related market dynamics" (Khalane, 2013, p.13). Following on from this statement are these observations: that one agile methodology, Scrum, is gaining popularity, and that it is becoming one of the most widely followed approaches in the software development industry (Szalvay, 2004; Nawaz & Malik, 2008). On the basis of these observations, a question 'how can high quality requirements be achieved in Scrum projects?' becomes both logical and urgent in this study.

However, a synopsis of quality assurance literature on agile methods reflects visible limitations in terms of the availability of insight on this subject in existing literature. References to these limitations include low magnitude, quality (in terms of publications platforms) and quantity of research publications in the field. In a review of empirical literature on agile software development methods, for example, Dingsoyr, Dyba and Abrahamsson (2008) found a limited number of works in the scientific literature. In this analysis, only 21% of all the papers were published in scientific journals, with the rest representing secondary research in non-scientific platforms such as magazines, opinion platforms and blogs (Dyba & Dingsoyr, 2008). Of major concern in this study however, is an apparent lack of literature on Scrum, with as few as 6% of the articles focusing on this methodology, and a similar percentage discussing quality assurance (ibid). Another concern is that quality and quality-assurance are discussed under a "broader agile umbrella" in the existing literature - as if all agile methods were altogether symmetrical (Khalane, 2013, p.15). In this observation, one

may be tempted to ask whether the imbalance in terms of research focus between Extreme Programing (XP) and Scrum can be justified. Questions can be asked like: Is this because there are similarities between the two methodologies? Is research on XP adequate to explain processes in Scrum? Despite similarities in the underlying principle of agile methods, the fact that each methodology has its own distinct principles of practice that warrant separate treatment, dismisses this argument as almost ludicrous (Maruping, Venkatesh & Agarwal, 2009). For example, whilst XP focuses on supporting implementation steps, "Scrum targets the planning and management of development projects" (Overhage & Schlauderer 2012, p.5453).

This point raises a concern, not so much in a realization that literature studies on agile methods tend to focus on XP methods, but that there is little or no attention given to management-oriented methods such as Scrum (Khalane, 2013). In other words, research to understand quality assurance practices in Scrum processes are necessary and urgent.

It is not only the numbers and type of publication platforms that matter, but also the variety of issues, such as the richness and quality of topics covered, that determine the adequacy of existing research on the subject. In terms of the number and depth of articles about the topical issues associated with quality assurance, the literature seems to be grossly inadequate.

2.1 Research Problem

Since Scrum is increasing in popularity and adoption in the software development industry, clarity, on related quality assurance issues in its processes, not only becomes important but also urgent. The problem however, is that despite the increase in its adoption and uptake, it is not clear how Scrum teams can maximize quality requirements in Scrum projects. Explanations are that Scrum is one of the most under-researched Software development approaches in agile methods. Because of this vacuum the risk of failure (remain realistic) but also unpredictable in Scrum projects.

3. Research Objective

The objective of this study therefore, was to explore the extent to which software quality assurance processes can be understood and applied to maximize the quality of software in Scrum projects. Quality in this context is defined as fitness for purpose and as the product meeting user expectations. The aim was to identify at least 5 Scrum projects with different teams and operating circumstances, so as to understand common quality assurance processes upon which Software Quality Assurance (SQA) inferences on Scrum can be

made. The goal was to contribute additional insight into quality assurance in Scrum, so as to contribute towards improved quality assurance practices in agile methods, generally.

4. Research Question/s

How can software quality assurance processes be understood and applied to maximize the quality of software in Scrum projects?

4.1 Sub-questions

1. What are the SQA processes applicable in Scrum projects?
2. What is the significance of SQA in Scrum projects?
3. How are SQA practices applied and enforced in development projects of the current organization/ team/s?
4. What is the SQA success rate in current Scrum development practices?
5. What are the determinants of the SQA success/ failure rate in Scrum projects?
6. How can software development teams maximize the quality of outputs in Scrum projects?
7. What are the challenges facing Scrum SQA?

5. Terminology of the thesis

Customer is an organization or a person who buys the product or service with the intention of using or offering a service to a third party.

Quality is defined as the completeness of characteristics of a product or service that satisfies or exceeds stated needs (Bevan, 1995; Fitzpatrick, 1996; Wilson & Hall, 1998).

Software Quality refers to the software's fitness for purpose or use (Alsultanny & Wohaishi, 2009; Jones & Bonsignour, 2012) the way in which the system conforms to defined requirements (English, 1996; (Sirshar & Arif, 2012) and how well it meets the customer's needs (Bevan, 1999; Galin, 2004; Berander, Damm, Erriksson, Gorschek, Henningsson, Jonsson, Kagstrom, Milicic, Martensson, Ronkko & Tomaszewski, 2005; Mnkandla & Dwolatzky, 2006; Yahaya, Deraman & Hamdan, 2008; Pressman, 2010; Nafees 2011; Sommerville, 2011).

Software Quality Assurance (SQA) refers to an effective and continuous quality validation as well as the verification processes within the software development life cycle. In terms of the definition of quality thus, SQA refers to the validation and verification of quality in the sense of ensuring the fitness for purpose, conformance to specifications, and that the end

product meets the needs of the customer (Mnkandla & Dwolatzky, 2006; Sommerville, 2011; Jones & Bonsignour, 2012; The Free Dictionary, 2013).

User is the person using the system after it has been fully developed and installed.

6. Structure of the thesis

The structure of the proposed research project is outlined in Figure 1.

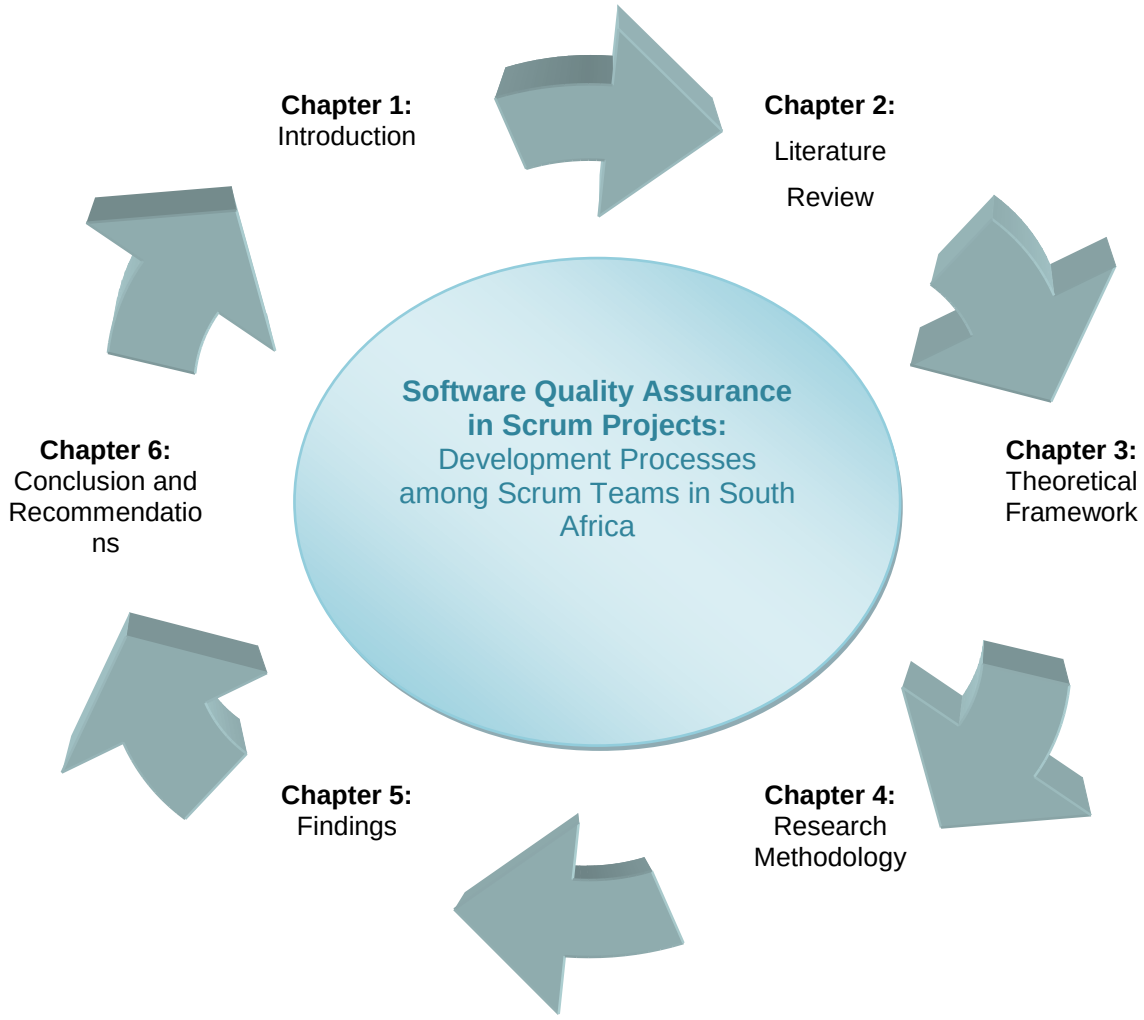

Figure 1: Thesis Structure

7. Conclusion to Chapter One

The aim of this chapter was to present the significance of IT in society and in business. It is evident that organizations rely on IT to operate effectively. The enormous increase in the use of IT in societies also substantiates the importance and criticality of developing quality products. It was also highlighted that though the increase in the use IT, particularly for managing business information, the IT industry is still faced with a challenge of developing quality products. Traditional methodologies are blamed for the high failure rate of IT systems as a result, organizations have adopted agile Scrum methodologies. However, despite the increase in the uptake of Scrum methodologies it is not clear how Scrum teams achieve the high quality that is required in Scrum projects. Therefore, the purpose of this study was to identify at least 5 Scrum projects, with different teams and operating circumstances, so as to identify and understand the common quality assurance practices adopted by software development teams within the Scrum methodologies and hopefully to contribute towards improved quality assurance practices in agile methods

The next chapter presents the literature on related studies, the emphasis being on the quality of software products, the importance of quality, and outlines different ways in which quality is understood and applied in software development processes.

CHAPTER TWO – SOFTWARE QUALITY ASSURANCE

2. Introduction

The previous chapter clearly highlighted the significance of IT in societies, communications and businesses as they rely on software systems for managing information effectively. It is on this basis that software quality and quality assurance becomes a priority. This chapter opens with a discussion of different views of quality. It presents testing as a validation and verification process of quality assurance.

This chapter is divided into four major sections: the first is a clarification of quality from a general perspective and of the importance of quality in software development. This is followed in 2.2 by a presentation of the different ways of ensuring quality in a software development process. In 2.3 there is a discussion of software development methodologies (traditional and agile) focusing on sequential phases of the development process, section 2.4 looks at related studies and finally the conclusion in section 2.5.

2.1 Software Quality

The phenomenon of quality, generally, is defined as the completeness of characteristics of a product or service that satisfies or exceeds stated needs (Bevan, 1995; Fitzpatrick, 1996; Wilson & Hall, 1998; Business Disctionary, 2013). For example, a product that is developed or a service that is rendered can be judged either as of high or low quality, depending on how it meets stated or non-stated standard requirements. In software development then, quality refers to the software's fitness for purpose or use (Alsultanny & Wohaishi, 2009; Jones & Bonsignour, 2012) the extent to which the system conforms to defined requirements (English, 1996; Sirshar & Arif, 2012) and meets a customer's needs (Bevan, 1999; Galin, 2004; Berander et al., 2005; Mnkandla & Dwolatzky, 2006; Yahaya et al., 2008; Pressman, 2010; Nafees, 2011; Sommerville, 2011).

2.1.1 Quality as Fitness for Purpose

In terms of 'fitness for purpose', a software should meet the intended goals of the customer in terms of scope, time and cost efficiencies (Elam, 2011). Software scope relates to the estimation and management of a customer's requirements (Pressman, 2010). Scope, time and cost relate to an important aspect of software development called project estimation. In the software development life cycle, there is always risk, either of not meeting the intended needs, by exceeding budget or of delivering poor quality software due to poor estimation

ending up wasting money (Sneed & Merey, 1985). Scope estimation, is where requirements are adequately articulated to ensure that the development process, and ultimately, the software product meet expectations, in terms of performance, without escalating time and costs (Bevan, 1997). Project scope is an important aspect that contributes towards achieving quality results in terms of products that are easy for customers to use and understand (Meli, 1999). From the development process point of view, quality as fitness for purpose also relates to the efficiency and effectiveness of a software product (Fitzpatrick, 1996). Further, it is not uncommon for customers to also perceive a product that is delivered on time and within budget, as being of higher quality (Khalaf & Al-Jedaiah, 2008). For example, if the size, time, effort, duration and cost of development process are accurately estimated, the risk of not meeting a user's requirements can be reduced (Pressman, 2010).

Quality as fitness for purpose therefore, includes efforts to ensure that development processes yield a software product that meets a customer's needs (Sallis, 1996). From the software's and customer's perspective this implies that it is not enough for a product to work, but it must also be reliable and easy to use (Kitchenham & Pfleeger, 1996). Unless the ease of use is clearly articulated, catered for and prioritized under the 'definition of requirements' phase, then, the usability aspect of the product may be threatened (ibid).

In terms of conformance to defined requirements, a software product should be flexible in its ability to adapt to new environments (Yahaya et al., 2008), provide value to customers and must meet the performance goals and the features defined in the requirements specification phase (Pressman, 2010).

A customer or user's needs and environments are continuously changing. Therefore, flexibility, including software customizability and adaptability, become an important aspect of quality. Flexibility is the degree to which the use of a software system can be modified easily during the software development life cycle and after the software has been released, as well as how well the software is able to adjust to changing requirements and needs (Eden & Mens, 2006; Truren, 2010). There may be a variety of aspects that can form 'system requirements' specifications. One of these is software flexibility which is either software process flexibility or product flexibility. Process flexibility focuses on ensuring that the business process changes are considered from the requirement specification phase up to the implementation phase. Product flexibility on the other hand refers to the requirement modification and the software's internal structure changes (Subramaniam & Zulzalil, 2012). As a quality component, flexibility is when a software product can accurately accommodate changes in a short period of time, and at a low costs.

In fact, most software users rate the customizability and adaptability to their evolving innovation needs as a major development requirement (Makabee, 2013), which is a significant quality aspect. Software should be customizable to reduce a user's costs. In this context customizability refers to the degree of ease with which software extends functionality in order to meet user's needs without altering the existing software and without accelerating maintenance costs (Advoss, 2014). An example of poor customizability would be when the addition of new features requires the software to be redeveloped, due to lack of adaptability (Subramanian & Chung, 2001).

The requirements and the operation environments often change. Hence, software adaptability to systems innovative changes is also critical for businesses to survive. In other words, it is not enough for a software system to only solve the problem but it has to accommodate changes, without causing failures in the existing system. Software adaptability is the degree to which the software is easy to adapt to external changes. It also refers to the capability of the software to accommodate modifications (Naik & Tripathy, 2008) in order to cope with major changes in business processes, without interrupting existing systems (Shen & Ren, 1990; Engel & Browning, 2006). In addition, adaptable systems are easily customizable by end users, and adaptability is important in reducing software costs by making it possible for a business to engage in new business opportunities, with minimum time delays.

As the dominant aspects of systems design requirements, compliance to software flexibility, which incorporates customizability and adaptability, implies compliance to a significant component of customer quality-expectations. For example, quality is ensured by complying with stated and unstated customer requirements. Thus, if flexibility expectations are not met, the customer will not be satisfied with the overall quality aspect of a software product (Pressman, 2010), even if other performance aspects are satisfactory. When followed and implemented properly, these attributes make it possible for customers to make changes or add new functionalities, easily, and without redesigning the entire software. It is in this context then, that compliance to these requirements, where they have been prioritized in initial specifications, is considered a significant aspect of quality in software development.

2.1.2 Quality as Meeting Customer Needs

In terms of meeting customer needs, quality means making customers' lives easier and more worthwhile by optimizing aspects that they care about (English, 1996). Meeting customers' needs also refers to a user's view point of quality, called quality in use. Quality in use is measured by assessing the software's effectiveness, productivity and the satisfaction of users when carrying out specific tasks (Bevan, 1999), as well as the capability of the

software product to help users achieve their goals easily (Nafees, 2011). On the other hand effectiveness is measured by the software's correctness and the accuracy achieved when performing specific goals (ibid). User satisfaction is usually measured by conducting a survey to find out how users think of a specific software product, by focusing on the software's Functionality, Usability, Performance, Reliability, Maintainability and the Service it provides. Usability is defined as the extent to which the software is easy to use, learn and understand (Bevan, 1999). Reliability is the extent to which a program can be expected to perform its intended function with required correctness and is able to maintain its level of performance (Naik & Tripathy, 2008; Pressman, 2010). Reliability is then determined by the number of defects found in the software (Serumgard, 1997). The software system is considered reliable if it continues to thoroughly perform well and achieve its goals over a long period of time (Fitzpatrick, 1996). Software reliability can be achieved by reducing complexity of the planning and development processes, making it easy to modify it when there are new requirements (Alsultanny & Wohaishi, 2009).

Software reliability also contributes to the fitness for purpose aspect of quality, in the same way as it informs the expectation of conformance to requirements in the manufacturing industry (Kitchenham & Pfleeger, 1996). From the manufacture's point of view in this instance, focus is on examining the product to ensure that the product is developed right the first time, and that it conforms strictly to specifications (Sirshar & Arif, 2012). Where quality is linked with the notion of value-for-money it is perceived as something that is determined by the amount of money that a customer is willing to pay for a product (Mnkandla & Dwolatzky, 2006; Naik & Tripathy, 2008; Pressman, 2010). Value based assessments are concerned with software design costs that alter when there are changes in the requirement specifications, and therefore a focus is often placed on limiting the cost of rework during the development lifecycle (Kitchenham & Pfleeger, 1996).

It is clear in this discussion that though quality has different categories of meanings, that the interests of the customer and the user are central to all interpretations. For this reason, it can be argued that quality should be the responsibility of every project stakeholder meaning that project stakeholders need to work together in order to deliver high quality software and to collectively address issues of poor quality by introducing techniques and controls for ensuring quality (quality assurance) in developed systems (Bevan, 1997).

2.2 Quality Assurance in Software Development

Software quality assurance has been in existence for a number of years. Quality assurance standards were first introduced in military contract software development in the 1970s where the purpose was to ensure that the development process conforms to defined quality

standards (Pressman, 2010). Due to the criticalities in today's software systems, software quality assurance practices are being given a high priority in the software development industry (Nawaz & Malik, 2008; Safecode, 2008).

Software Quality Assurance (SQA) is defined as an effective and continuous quality validation which includes the verification processes within the software development life cycle. In terms of the definition of quality thus, SQA refers to the validation and verification of quality in three senses: that of ensuring the fitness for purpose, that of conforming to specifications, and that of the end product meeting the needs of the customer (Mnkandla & Dwolatzky, 2006; Sommerville, 2011; Jones & Bonsignour, 2012; The Free Dictionary, 2013).

2.2.1 Quality Assurance as Process Validation and Verification

Validation refers to the process where a component within the development process is checked for consistency and completeness, to ensure that the software meets the specified requirements and that the end product is fit for purpose (Galin, 2004; Easterbrook, 2010). This entails the clarification of consistency and correctness in all the phases of the development life cycle (Naik & Tripathy, 2008). Depending on the methodology applied in the development process, the validation process can be conducted during the early stages of the development process, at the end of development phase or throughout the software development life cycle (Khraiwesh & Jordan, 2011). In this process, emphasis is placed on modelling, prototyping and user evaluation, to ensure that the development team develops the right product. The aim of validation is to ensure the usability and usefulness of software, by finding and fixing defects. It is also to strengthen the development process through functionality evaluations, to ensure that the development phase satisfies the conditions imposed at the beginning of that phase (Massey & Satao, 2012 Sommerville, 2011). For this reason, the validation (also known as the verification) process plays a major role in improving quality in software development (Khraiwesh & Jordan, 2011).

Validation is concerned with enforcing high quality in systems, by testing and inspecting to ensure that the end-product is as near-bug free as is technically possible (Easterbrook 2010).

2.2.2 Software Testing as the Validation Process

Software testing is a process of executing a program in order to find defects and to verify that the software does what it is supposed to do (Khraiwesh & Jordan, 2011; Tuteja & Dubey, 2012). Defects are the common errors encountered while developing the software,

they can be the bugs in the code or a system that cannot work according to expectations (Mosaicinc, 2001).

Testing is an important SQA process in that it validates and ensures functionality compliances in respective phase/s of the software development life cycle. Testing is used to reduce bugs and carry out related fixes. In fact, the cost of defects found during the testing phase may be less than that associated with those found after the software is released (Naik & Tripathy, 2008). In this case the significance of testing is to minimize the amount of rework and costs associated with bug fixes. Testing also helps to improve the conformance to the requirements' specification.

Inspection, which is a component of testing, is equally important in that it facilitates the identification of defects that can then be retested and fixed (Wallin, 2002). The objective of both testing and inspection is to produce high quality software, by ensuring that the software has a low number of defects, and is free from vulnerabilities (Webopedia, 2014). Testing is also to ensure that the product reaches and complies with the required standards of maintainability, reliability as well as reusability (Fitzpatrick, 1996; Sirshar & Arif, 2012), thus making sure that the software is fit for its purpose.

The SQA can also be viewed as compliance with planned standards (Pressman, 2010) which monitor technical requirements, to ensure conformance to specification requirements defined in the technical requirements (Galin, 2004; Ahamed, 2011). Quality standards are further divided into product and process standards. Product standards include requirement documents and documentation standards which are used by developers as a guide on how programming languages should be used. Process standards on the other hand define processes that should be used while developing the software product. These standards include document description, definition of specifications and the good development practice. In this instance the significance of software standards is to define the level of quality to be achieved in order to satisfy customer needs, for example (product) dependability, usability and performance (Sommerville, 2011). Standards help to keep consistency throughout the software life cycle, they increase efficiency, which shortens development time. Standards stress the importance of meeting minimum projections and of planning processes according to pre-defined quality projections. Furthermore, it is the responsibility of the project manager and the quality manager to plan these standards early when starting the project. Planning involves the prioritization of standards by specifying those standards that can be modified and those that cannot (be modified). This is because technology and programming languages change frequently, therefore software standards need to adapt easily to changes.

Software Quality Assurance (SQA) consists of a set of quality models and assurance processes which cover all phases of the software development life cycle. The aim of these quality models is to identify factors which affect software quality. These processes are very important as they help in building and maintaining the quality of the product (Tomar & Thakare, 2012). One component of the software quality assurance process is process control, which ensures control in the software development life cycle. Process control is one of the effective mechanisms which ensure the product's fitness for use (NASA, 2004). It is used in combination with software cases in design and development of the software system. Assurance cases are the collection of sets of bodies of evidence organized as arguments that demonstrate functionality, safety, security and quality of a software product (Conklin, 2011).

2.3 Software Development Methodologies

Though the intention of every software engineering process is to produce the best end product in terms of quality, success is largely dependent on a sound balance between key process factors within an appropriate development methodology (Aggarwal & Sigh, 2001). In this respect, software developers often have to decide whether to use traditional methods, such as waterfall, or agile methods, such as extreme programming (XP) or Scrum, in specific software engineering projects (Geambasu, Jianu, Jianu & Gavrila, 2011). Obviously, arguments on quality assurance aspects of software vary, with notable differences in perspectives prevailing between the proponents of traditional and of agile methods, on what are the ideal practices. Disagreements often revolve around the merits and demerits of each methodology in their respective contexts (Sousa, 2013).

2.3.1 Traditional Software Development Methods

Traditional methods such as spiral model, rapid prototyping, incremental development, V-Model and Waterfall model are unique in the sense that they are based on a sequential series of steps (Rohil, 2012; Leau, Loo, Tham & Tan, 2012). The Waterfall model, which was developed by Royce in 1970s (Smith, 2003) with the purpose of introducing a level of formality into software systems development, is a useful example of traditional methods. It is characterized by a clear articulation of development requirements that inform each of the phases in the development process (Khalaf & Al-Jedaiah, 2008). In other words it is a highly structured, sequential software development process that progresses through various software design and development phases in a linear fashion, like a waterfall (Cadle & Yeates, 2008; Czarnacka-Chrobot, 2010; Select Business Solutions, 2013). The merit of the model is largely on the interdependence of sequences between all phases. In this respect,

the end of each phase becomes the start of the next phase and the output of each phase becomes the input of the following phase (Cadle & Yeates, 2008).
The significance of this approach is that compliance with development requirements is ensured at the end of each phase, thereby limiting the margin of error at the output phase. In other words, each phase has a deadline and once a phase is completed, it cannot be revisited (Mccormick, 2012). Requirement analysis which includes the investigation and understanding of users' business processes, constraints, risks and performance levels, is prioritized in this model (Khalaf & Al-Jedaiah, 2008).

The process phases in the waterfall model consist of requirement specification, system design, coding, software deployment and the system maintenance (The SDLC, 2005; Tuteja & Dubey, 2012). As reflected in section 2.2, quality assurance is a significant aspect of all the software development phases, in the waterfall model.

2.3.2 Requirement Specification Phase

The first important phase in the development life cycle, critical to the success of the project (Bender RBT Inc, 2003) is the requirements gathering and analysis, known as requirements specification, which is done by system analysts (Smith, 2003). The aim of this process is to gather as much information about the system as possible. The aim is also to verify and validate such requirements in order to limit errors, and avoid changes later in the development process, thus improving quality (Young, 2002).

In the requirements specification phase, system requirements are divided into functional and non-functional aspects. Functional requirements are features and specifications that define what is going to be built by the developers (Szalvay, 2004). Non-functional requirements on the other hand, deal with the quality issues of the system, including security aspects (Chung & do Prado, 2009). Verification and validation, which are central elements of the quality assurance (QA) process, feature strongly in this phase (Massey & Satao, 2012). In this phase, requirements are ascertained, then accurately verified and validated using peer reviews, scenarios and walk-throughs in order to satisfy project stakeholders (including the client, developer and the user). The verification and validation aspects are the distinct strengths of the structured models in that they improve clarity which helps inform QA and control in all other development phases (Galin, 2004).

The requirements specification document, which becomes the input of the design phase and the user requirement document, is an important aspect of this phase. It is used by system[s] analysts to communicate their understanding of the system with users. Meanwhile, the user requirements document describes the system's security requirements, physical, interface, performance, data and other important aspects of the software development process

(OSQA, 2009). These two documents help developers to develop high quality reliable software that meet users' needs on time and within budget (Young, 2002).

The importance of the requirements gathering phase therefore, is that it helps developers to accurately map the requirements to a subset of the module, thereby increasing the correctness of the system. The significance of this phase is that it also helps to estimate the development timeframe, to predict possible problems and to alleviate associated risks in the development process. This way, appropriate plans can be made to ensure that the end product is of high quality (Leau et al., 2012).

On this basis, it is also important for project stakeholders to understand and work together while analysing the requirements. When all the project stakeholders work together as a team with a common goal, it facilitates the collaborative effort towards the implementation of software quality control measures (Owens & Khazanchi, 2009).

Within the waterfall methodology then, it is only when all the project stakeholders are satisfied with the requirements specification that the phase gets signed off, in order to move on to the second phase. It is clear that the requirements specification phase is the most important phase in the Software Development Life Cycle (SDLC) in the sense that all other phases are dependent on the success of this phase. The documents created in this phase are used, not only as a guide in the software design phase, but also in the rest of the phases throughout the entire development process.

2.3.3 The Software Design Phase

The design phase defines the structure and process to be followed in the development (coding) phase (Smith, 2003). Appropriate methods are outlined in this phase, including interviews, workshops or prototype development (The SDLC, 2005). Information gathered in the previous phase is translated into system design documents which accurately describe the design of the system. These documents are used by developers as a guide to the coding phase (Owens & Khazanchi, 2009). After customer verification, the documents can be used to guide further development processes and related system modifications.

Emphasis on QA is also evident in this phase. For example, during the information translation process, data capturing procedures are accurately followed to ensure data correctness in the system (Jirava, 2004). In line with quality assurance priorities, the SQA staff is required to ensure that planned activities are carried out as planned, with regular reviews throughout the development process to avoid design complexity (BSSC, 1995). The importance of these reviews is to improve the quality of documents developed and to simplify the development phase. Detailed software features, including functional hierarchy

diagrams, business rules, business process diagrams, entity relationship diagrams, an access control matrix and an architectural design, are also developed, with constant verifications to ensure quality compliance (Leau et al., 2012). The objective of the architectural design is to develop an understandable, maintainable and scalable system (Liong, 2004; Khalaf & Al-Jedaiah, 2008).

Compliance with design standards is another aspect of quality assurance that is also prioritized in the design phase (Naik & Tripathy, 2008). In addition to customer specifications, the software quality assurance team ensures that sought software designs adhere to the organization's software quality standards in this phase (Owens & Khazanchi, 2009).

The successful completion of this phase includes the transformation of system requirements into detailed specifications covering all aspects of the developed software. Once the development team is satisfied with the architectural designs and plan the project get passed to the coding phase in order for developers to start writing code, so as to reach project goals.

The output of this software design phase includes a detailed design, well defined functional modules and a clear process towards the development of a quality software product. Emphasis on constant validation and verification of design aspects and adherence to software design standards informs the next phase of the SDLC (the coding phase), with a positive impact on the development process and the resultant product. Association of this practice with waterfall methods also suggests the centrality of quality assurance practices in traditional methodologies.

2.3.4 The Coding Phase

After the finalization of the design phase, the developers start with the actual development of the software in the coding phase (The SDLC, 2005). Within the waterfall methods, the coding (implementation) phase of the SDLC deals with the development of small programs called units. As in other phases, units also get tested constantly to make sure that they work according to defined requirements, that they comply with design and coding standards and that they are error-free (OSQA, 2009). In terms of QA, tests include writing unit test and integration tests to ensure that units work when combined together (Smith, 2003). The aim of writing a unit test is to ensure that each unit of program meets customer expectations, that the software works after integrating it into the existing software, and that defects are detected and removed (Ichu & Nemani, 2011). Defect density is commonly used to measure software quality. In this phase the defect density is reduced by constant code reviews where each developer code is reviewed for quality.

Code reviews are performed by peers to simplify code readability, to ensure easy maintenance and to improve the overall quality of the product. These reviews help to identify defects by ensuring that coding standards are followed through the development process (Kemerer, 2009). In this way, development costs and time during the product development and in the maintenance phase can be reduced.

Emphasis on QA through coding standards is also evident in this phase. In order to ensure a high quality end product, the software is developed following coding standards which are constantly reviewed for compliance (Parasoft, 2010). The purpose of these standards is to define how code should be written in order to help developers write simple reusable software. Studies show that codes that are written following standards are easy to use on other technologies (ibid). Therefore complying with coding standards can ensure high quality software.

Although it is important to have QA measures in place in the previous phases, it is in this phase that it is of the greatest importance: the developers also have to strive to produce near bug free code, because at this stage it is still relatively inexpensive to fix errors. Therefore, even though the emphasis of this phase is on writing code, the development of unit tests also plays a major role not only in making sure that each unit works but also to simplify the testing phase, which is conducted next.

2.3.5 The Testing phase

The testing phase deals with the verification of software to ensure that it meets defined requirements, and that it is fit for purpose (Naik & Tripathy, 2008; Petersen, Wohlin & Baca, 2009). The testing process includes integrating testing, system testing and acceptance testing.

During testing, QA analysts thoroughly test the software with the intention of finding errors and assessing functionality. Identified errors are corrected in order to ensure that the product is near bug free (Galin, 2004; Owens & Khazanchi, 2009; Balaji, 2012).

Although testing is only conducted after the coding phase, it contributes strongly towards quality assurance in waterfall models. However, not all errors are detected in this phase, hence the significance of bug fixes in other phases such as the maintenance phase. The maintenance phase which is discussed next deals with additional requirements, bug fixes and software upgrades.

2.3.6 The Maintenance Phase

It is every developer's aim to produce error-free software. However, there are errors that are not detected until the software is released. The maintenance phase deals with finding and fixing errors undetected while testing the system and with providing software upgrades (Jirava, 2004). This phase focuses on modifying software to work with new or different computing platforms.

When doing software maintenance, if code is difficult to understand, too much time is spent on reading. Reduced readability leads to more time spent reading code which then results in higher maintenance costs. A study conducted by Collar and Valerdi (2006) shows that improved code readability leads to less time spent reading it. Less time reading code results in lower costs throughout the software development phases of the life cycle. It is recommended that developers write simple code that is easy to understand because code that is easy to read should be easier to modify. The significance of software maintenance is to keep the business operation running at all times (Smith, 2003).

The waterfall model is considered the most reliable methodology in terms of quality assurance, and affords numerous advantages for users in terms of stability in its resultant well documented systems (ibid). Despite the success of the waterfall methods in large and complex projects, a number of drawbacks when it has been used in smaller projects have been reported. These include inflexibility to changing requirements, inability to revisit closed phase(s), assuming that all the requirements can be gathered in one phase (requirement specification) and limited interaction between the customer and the development team (Tayntor, 2002; Khalaf & Al-Jedaiah, 2008). Inflexible and time consuming development processes, together with an inability to respond to changing requirements had caused organizations to use other methodologies (Huo et al., 2004). In particular, because of the concerns over inflexibility in requirement changes, long development time and budget overruns, the software industry experts gathered in 2001 to explore alternative methods (Knippers, 2011; Leau et al., 2012).

The less-structured (almost unstructured), flexible alternative known as agile software development methods were introduced, and have been adopted on an increasing scale by IT organizations in the 21st century (Ambler, 2005).

2.3.7 Agile Software Development Methodology

The agile methodology is a group of human orientated adaptive and flexible software development methods (Timperi, 2004), with a focus on iterative and incremental development - the product is designed, implemented and tested within a shorter space of

time than would be possible using one of the traditional methodologies (Balaji, 2012; Kulas, 2012). Flexibility refers to the ability to adapt to changes during and after the development (Innolance, 2013) through creative and incremental processes, easily and quickly (Knippers, 2011; Mansor, Yahya, & Arshad, 2011; Mohammad, 2013; TechTarget, 2013). In this methodology, emphasis is placed on close collaboration between self-organizing and cross-functional teams that allow rapid delivery of software, to meet customer needs (Moniruzzaman & Hossain, 2013). Agile methods enable changes to be conducted at any stage of the development life cycle and can be applied to any software process (Knippers, 2011). The main purpose, therefore, is more about improving turnover and increasing output, rather than being over-obsessed with rigid process routines. The aim is to develop an iterative and incremental software product that provides customer satisfaction, within a shorter development life cycle, with a reduced bug rate, and most significantly, to allow changes at any phase during the development process (Leau et al., 2012).

Unlike waterfall methods, which assume that requirements can be gathered in one phase, agile software development methodology is based on understanding that requirements are dynamic, meaning that they can change at any time during the development process (Moniruzzaman & Hossain, 2013). The ability to respond rapidly to customer needs while providing good quality software has resulted in a growing adoption of this methodology in the software development industry (Bhasin, 2012). A common ingredient in this process has been a close business relationship with frequent feedback between the development team and the customers (Huo et al., 2004). Further, agile methods divide the SDLC into small increments or iterations (Leau et al., 2012). These are developed by small teams following continuous design and testing based on rapid feedback and response to change. When iteration is finished, it gets delivered to the customer even if the entire software is not completed (Moniruzzaman & Hossain, 2013). After the final iteration, completed modules are presented to the customer for review.

The advantage of module review by the customer is early identification of, missing functionalities, areas that need to be changed and whether or not completed modules meet the customer's expectations. In this process, errors detected in one module are easier and less-costly to correct than those found only when the software is integrated (Andersson, 2003).

Agile methodologies incorporate methods such as Extreme Programming (XP), Crystal clear Methods, Test Driven Development (TDD), feature-driven development, adaptive software development methods, and Scrum (Ahmed, Ahmad, Ehsan, Mirza, & Sarwar, 2010; Nawaz & Malik; 2008). With innovative diversions from the rigid routines in the agile alternative, quality assurance aspects remain a highly contested terrain. For example, the agile

Manifesto which is the agility blue-print, redefines the software quality assurance (SQA) and related processes (Fowler & Highsmith, 2001) with notable omissions of some of the trusted traditional routines (Sfetsos & Stamelos, 2010). Arguments against this agile practice are that it nullifies some of the conventional quality assurance roles and responsibilities, replacing them with chaos – much to the dismay of the proponents of traditional methods. Despite this controversy, agile methods continue to grow, both in popularity and in rate of adoption, over a wide cross-section of the software development industry (Winter, Ronkko, Hotchkiss & Ahlberg, 2008).

Of particular interested in this study however, has been the increasing uptake of the Scrum methods (Dingsoyr, Nerur, Balijepally & Moe, 2012; Khalane, 2013). In support of this point, Kayes (2011) declares Scrum to be one of the most commonly used methodologies in project management practices (Khalane, 2013). Whilst the scientific community appears to appreciate the significance of the software quality assurance debate, with a growing trail of research efforts, there seem, unfortunately, to be more publications on traditional methods than there are on agile methods. The modest number of publications dealing with agile methods also tends to be focused on methods such as XP and TDD, with inadequate research in the literature on Scrum methods in general and on quality assurance in Scrum in particular (Timperi, 2004; Sfetsos & Stamelos, 2010). Thus, it can be concluded that, apart from the case study by Schwaber (2004, p.13), little is known in practice “about how Scrum teams achieve software quality assurance”.

In this study, this situation is seen as a major concern for the users of Scrum, which is a methodology whose use is growing in the IT industry.

2.3.7.1 The Scrum Methodology

Scrum methodology is an effective iterative and incremental project management framework which facilitates agile software development (Santamaria, 2007) in complex environments (Rising & Janoff, 2000). It attempts to control risks of unpredictable changes within the software development process and improve communication among project stakeholders (CS, 1995, Ahmed, Ahmad, Ehsan, Mirza & Sarwar, 2010). It guides project management on how a development team can function effectively in order to achieve product flexibility in environments where requirements change frequently (Nawaz & Malik, 2008). In this method communication is facilitated by means of short day-to-day stand-up meetings which are usually conducted by team members first thing in the morning (ibid). The Scrum meetings are coordinated by the Scrum master (team leader) with the purpose of monitoring team progress as well as identifying impediments. During the Scrum meeting three questions are answered by team members (Rising & Janoff, 2000; Sutherland, 2004; Mohammad, 2013):

- What did you do yesterday?
- What will you do today?
- Are there any impediments in your way?

These kinds of meetings also help in maximizing the cooperation between teams, sharing of knowledge among team members as well as improving productivity and quality of the product being developed (Ahmed et al., 2010). The significance of frequent communications in Scrum methodology is to help the development process to adapt easily to changes in priorities (ibid).

Scrum methods are regarded as flexible in that they can be used for both small and large projects. This aspect improves the management of a project, since a project can be broken down into smaller modules which can be finished within a short period of time. The study conducted by Ahmed et al., (2010) shows that among the common agile methods, Scrum was the most commonly used methodology. The purpose of the study was to investigate the impact of Scrum methodologies on productivity and quality. The results showed that industry is satisfied with the productivity and improved quality provided by Scrum methodologies.

The focus of Scrum (development) is on the management aspect of the development. Scrum methods consist of the development iterations called sprints. A sprint represents an iteration that is delivered within a period of one month (Nawaz & Malik, 2008). The significance of sprints is to coordinate the changing demands and requirements as well as to prioritizing the product's requirements (Timperi, 2004). This procedure prevents leaving the introduction of requirement changes until the end of the iteration, as well as providing a frequent customer feedback on project costs (Mnkandla & Dwolatzky, 2006). The frequent feedback helps developers to better understand the system being developed. Understanding the system helps in improving the development team productivity and the usability of a product (Sirshar & Arif, 2012). This approach provides flexibility and produces a system that is responsive to additional requirements discovered late in the development process (Schwaber, 1994). The Scrum process is shown in Figure 2 below.

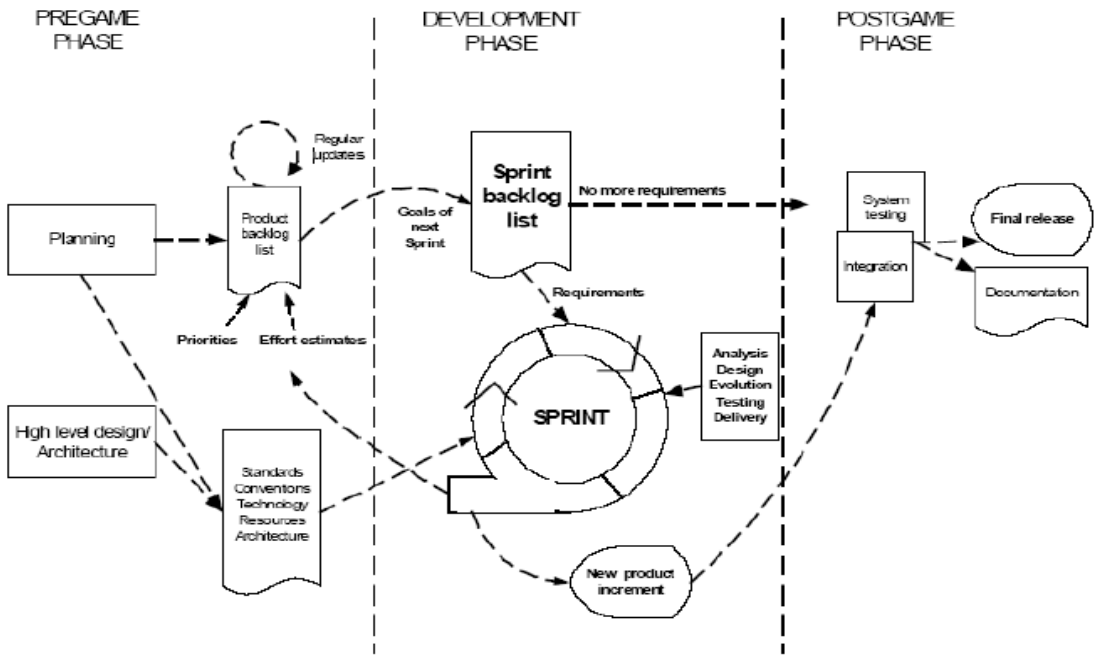

Figure 2: The Scrum process (Nawaz & Malik, 2008)

As indicated in Figure 2, Scrum development follows an approach which consists of three phases: the planning phase, the pregame phase and the development phase (Schwaber, 1994). The planning phase focuses mainly on project planning and the development of architecture and consists of defined processes (Schwaber, 1994). The development team develops an architecture which can be modified at a later stage if needs be (Rising & Janoff, 2000). The purpose of architecture is to ensure consistency throughout the development process. It deals with the definition of releases on a specified sprint as well as the estimation of schedules and costs. Costs include cost per activity which is estimated and verified while planning the backlog (CS, 1995). Backlog includes all the features that have to be implemented in the developed software (Imreh & Raisinghani, 2011) and consists of one or more releases.

The difference between Scrum and other methodologies is that, Scrum takes into consideration the fact that analysis, design and development processes are unpredictable. Thus a control mechanism is used to manage the unpredictability and control risk. The unpredictable aspects in the software development process include requirements, resources, time and technology (Nawaz & Malik, 2008). The end results of this mechanism are responsiveness and reliability. Scrum helps organizations to change project deliverables at any time during the development process (Schwaber, 1994).

Crystal Clear is one of the recent agile development methods which focuses on people and it can be applied to small projects with groups of fewer than ten developers. Test Driven Development is an agile method where tests are first written and the development is done in iteration on the basis of the tests (Sirshar & Arif, 2012).

With agile methodology, an organization can deliver a usable product on time within a specified budget and meet the needs of the customer. However, the most frequently asked question with regards to implementing agile methods is "How can agile methods ensure product quality when a product is delivered within such a short time period?" With the high failure rate experienced in IT projects this question is worth a thorough investigation. Hence Huo et al., (2004) conducted a study to investigate whether agile methodology can ensure that the software it delivers is of a quality comparable to that that would be delivered by a waterfall methodology. Several studies demonstrate that agile can improve software quality (Schwaber, 1994).

It is on this basis that software quality and quality-assurance in the agile development methods was chosen as the subject of analysis in this study.

2.4 The Scope of SQA Research Coverage

An analysis of the key topical issues covered in major journals on agile methods in general between 2003 and 2011 only focus on adoption, control, flexibility, and distributed environments (Lyytinen & Ngwenyama, 1992). However, the focus of this study is on understanding exactly how software quality can be achieved (and maximized) through the use of agile methods. Adoption only addresses matters of uptake, preference and use, rather than the quality assurance aspects (Nerur, Mahapatra & Mangalaraj, 2005). Similarly, flexibility tends to focus on time-efficiency and ease of adaptation to changes, more than it talks to quality assurance processes (Smith, 2003). Unfortunately, the area of quality assurance in agile methods is minimally addressed, with Scrum almost entirely non-existent in the literature (Abrahamsson, Conboy & Wang, 2009; Khalane, 2013).

In response to this limitation, Khalane (2013) undertook an investigation into the quality assurance process in Scrum projects, with a focus on how Scrum methods facilitate quality assurance in the form of ‘meeting user expectations’. Using a case study in one organization Khalane (2013) found a complete lack of concrete guiding principles in the Scrum process. It was also found that due to a lack of guiding principles, the analysis of quality assurance processes in Scrum projects can only be conducted and understood in a case-by-case (case specific) context. At the same time limitations in solid user representation and a lack of dedicated testing were found, and these were linked to an inability to meet user expectations. In this case study, it was argued that the situation could be different in another

context. Issues such as fluctuations in capacity demands, testing and quality as well as coding and testing expertise are cited, as context dependent variables, to argue for the context customization of Scrum. In the conclusion, it is argued that Scrum should be viewed as a “framework of empty buckets which need to be filled with situation specific SQA practices and processes” (Khalane, 2013). In view of the background of limitations in SQA, and in the light of these findings, it is clear that understanding and articulating quality assurance processes in Scrum is complex. Given a lack of guidelines, arguing for more case specific investigations as a basis through which a repertoire of inferential insight becomes logical, is necessary and urgent.

In order for software developing companies to deliver quality software on time and within budget, it is recommended that they first define the software process to be followed through the development life cycle. This has to be done in collaboration with the customer of the software being developed. Software quality techniques, initiatives or proven software engineering alone are not enough to ensure that organizations produce high quality software (Khalane, 2013).

2.5 Conclusion

This chapter provided a background to the research problem outlined briefly in the previous chapter, the emphasis was on the quality of software products. It began with a discussion on different ways in which quality is understood in the software development process. Quality assurance in software development was outlined in section 2.2 where testing as a validation and verification process was identified as an important process which contributes to improving the quality of software systems. The scope of software quality assurance was identified as a concern. In this respect, it was stated that there are few studies focusing on improving quality in agile methods. This discussion was followed by a detailed discussion of quality assurance in traditional and agile methodologies.

The next chapter focuses on the theoretical underpinning adopted in this study, by exploring various theories in software engineering and their importance in the software development process.

CHAPTER THREE – THEORETICAL UNDERPINNINGS

3. Introduction

The term theory (philosophy of knowledge) refers to clearly and validated systematic explanations of the natural world such as facts, laws, and inferences as well as tested and confirmed hypotheses (NCSE, 2012; Pettigrew & McKechnie, 2001). A theory attempts to understand real world occurrences or phenomena through explanations and predictions of natural phenomena, behaviour or occurrence (Creswell, 2003). Theories are found in almost all disciplines, including social science fields such as sociology, anthropology, education, economics, natural sciences and multidisciplinary fields such as information systems (ibid). They "appear as arguments, discussion or rationale" or "interpretations and explanations of phenomena" which are based on assumptions (Current Nursing, 2013).

Theory consists of a set of interrelated constructs formed into propositions, concepts, definitions (Coreil, 2010; eSource Research, 2014). In this instance propositions are used to predict a phenomenon by specifying relations between variables for explaining natural phenomena (Creswell, 2003). There are various definitions of theory that differ from discipline to discipline and can be used for a variety of reasons in a multiplicity of forms. For example in a non-natural science discipline theory refers to "unproven, [untested] or speculative phenomenon" (Pettigrew & McKechnie, 2001).

This chapter presents the theoretical underpinning to this study, in particular Structuration Theory (ST), to explore software quality assurance in a Scrum project and in software development processes among Scrum teams. The chapter introduces the use of theories in the IS discipline, and relates to how ST is applied in similar studies as well as the relevance of ST in the current study.

The chapter is divided into seven sections: the introduction is in section 3 followed by a discussion on the uses of theories in research in 3.1. This is followed by a discussion of possible theories in software engineering in section 3.2, a motivation for the adoption of structuration theory in section 3.3, the relevance and use of the structuration theory in section 3.4, the application of ST in the current study in 3.5 and a conclusion of the chapter in section 3.6.

3.1 The uses of Theories in Research

Theories can be used for various reasons including establishing new inventions (Zimmerman, 2012), attempting to develop effective ways to influence, to experiment, as

well as to explain and change behaviours (eSource Research, 2014). According to Gregor (2002) people invent theories in a form of concepts (which corresponds to the real world), models, and schemes. These forms of concepts can then be used to make sense of experience and modify constructions (knowledge) in the light of new experience (ibid). In research, theories help researchers to organize, communicate and simplify complexities of the social or natural world with the intent of building knowledge through deductive or inductive approaches (Reeves, Albert, Kuper & Hodges, 2008). These approaches are discussed in the section that follows. A theory also provides a conceptual understanding of aspects pertaining to how societies work, organizations operate, and why people interact in a certain way (ibid). For example, phenomenology can help researchers to explore how individuals make sense of the world by looking at subjective experiences. In other words a theory acts as guide in acquiring and formulating new knowledge.

There are various distinct ways in which theory is used in interpretive studies (Gay & Weaver, 2011). These include using a theory as a guide to design an iterative process to collect and analyse data which in the end produces a research product (Walsham, 1995). In this respect, theories also provide a framework for analysis, models, an efficient method for conducting field work and clear explanations of the world (Udo-akang, 2012). Because there are various theories which exist, researchers categorize theories based on their functions, purpose, limitations and goals (ibid). For instance researchers tend to use typologies as a basis for the classification of theories by identifying types which include the hypothetical-deductive method, inductive grounded theory, meta-analytical theory, social construction theory and case-study theory, for describing the purpose, definition, boundaries and goal theories (Gay & Weaver, 2011; Udo-akang, 2012).

In this instance the term typology refers to the “study and classification of language according to structural features, especially patterns of phonology, morphology and syntax without reference to their histories” (Dictionary, 2014, p.1). Typology is a type of conceptual framework which significantly helps to formulate research goals especially when using mixed methods, as it provides a common language used in a particular field (Teddlie, 2006). Typologies include methodological design such as qualitative and quantitative methods but also feature a combination of these methods, (also) known as mixed methods. Typology may be used in various disciplines within the social science spectrum for the clarification of phenomena in terms of the common characteristics among phenomena (Mouton, 1996).

Similarly, a model is a set of logical and mathematical relationships between variables which represent a situation in a research study (Jensen, 2004). In this respect the emphasis is on describing relationships between variables (ibid). Both typology and models work towards the formulation of a conceptual framework. A framework refers to “a collection of models and

studies gathered from literature" (Hussey & Hussey, 1997). It can be used to describe and explain phenomena and a guide to the positioning of the study, thus limiting the examined field of study or topic and generalizing ideas for further research (Udo-akang, 2012). From this statement it is evident that a theory has numerous functions which are applied to studies in whatever way is appropriate. Therefore, the choice of a theory is based on the research question, assumptions or goals and the objectives of the study (ibid). For example, considering the nature of the current study, the data to be collected to answer the research question requires a direct interaction with industry experienced experts and academics, who have been involved in research within the Information System (IS) discipline and who have practical experience in software development methodologies. The main aim is therefore to get rich qualitative data which will be described and interpreted with the intention of obtaining an understanding of quality assurance processes which exists under Scrum methods. The collected data will then be analysed and interpreted, where the key facts (in the form of themes, pattern and codes) will be identified. The patterns revealed in the analysis will be used to produce a theory. This theory will be used to provide guidelines for teams operating under Scrum methods to maximize and improve product quality (both internal and external).

Theories are derived from or can be used in two important methods of scientific reasoning, deductive and inductive (Gotelli & Llison, 2004; Udo-akang, 2012). The in-depth clarification of how theories are used under each approach, and ultimately the clarification of an adopted approach is outlined in the sections that follow.

3.1.1 Deductive Reasoning

Deductive reasoning entails universal laws of cause and effect which assume that reality consists of objectivity defined facts. It begins with the exploration of the existing theories upon which a hypothesis maybe developed, prior to observations from the real world (the gathering of evidence), followed by the testing of the hypothesis, a critical interpretation and a inferences on the findings (Gotelli & Llison, 2004). Deductive reasoning is often aligned with a quantitative methodology where theories are used as an organizing model for research questions or hypotheses, data collection procedures and as a framework for the entire study (Creswell, 2003). Surveys and questionnaires are some of the quantitative methods where deductive reasoning is commonly applied. Theories are also used as a predictive mechanism against which observations can be tested, confirmed or rejected. For example mathematical postulations (theorems) are used to construct, test and solve numerical problems.

The aim of the current study was to understand quality assurance practices adopted by software development teams within the Scrum methodologies. This process requires an in-depth understanding of procedures and motivations of these methodologies, as well as of related implications and the (underlying) impact on final outputs. The bulk of this insight requires in-depth descriptions and explanations of choices and procedures – which requires qualitative rather that quantitative accounts of this research phenomenon. Therefore, the deductive approach would have been inappropriate. Instead, the inductive use of theory was followed.

3.1.2 Inductive Reasoning

Inductive reasoning is a bottom-up approach to the use of theories in research, in that it begins with an observation, identifies patterns and inferences and then develops a single hypothesis, before explaining it to formulate a theory (Gotelli & Llison, 2004). Among other formats, Grounded Theory (GT) offers a clear example of an inductive approach to research where a theory may be generated at the end of the study (Creswell, 2003). In inductive reasoning, the researcher begins by gathering data often using qualitative methods which include observation and interviews where open-ended questions are asked from participants. The collected data is then analysed where themes and patterns are identified to form categories (ibid). The researcher then focuses on identifying pattern generalizations or themes whilst making inferences to theories from literature (Creswell, 2003). Under this approach, a theory can be used to explain behaviours or attitudes. The inductive approach can also be used as a lens to shape the research process and the questions asked, and as an analytical guide to data analysis, offering a close link between data and theory (Gotelli & Llison, 2004). When placed at the beginning of the study it is used as a lens that shapes questions asked (ibid).

Of major focus in the current study were the qualitative accounts of motivations and related issues in quality assurances Scrum projects. Obviously, established practices behind preferred methodologies serve as a basis (perhaps, a theoretical basis) upon which quality assurances are viewed in software development operational traditions. Nevertheless, the aim of the study was to go beyond fixed notions in preconceived motivations, in order to gain deep and perhaps inter-subjects dynamics of quality assurance practices by teams in Scrum projects. In this quest, a theoretical framework to offer a lens upon which a holistic investigation of this process could be built became both necessary and urgent.

Given that the field of software development is largely technical (software-engineering), albeit with a human element, the relevance of selected software engineering theories is also explored in (section 3.2).

3.2 Software Engineering Theories

Software development is the discipline (within software engineering), a large part of which pertains to the development and maintenance of software systems. Drawing on a number of Software Engineering (SE) theories, software development attempts to develop reliable and efficient systems which meet defined customer requirements (Al-azzah & Yhya, 2011). The core theories in SE include the Complexity Theory, Complex Adaptive Systems, the Theory of Boundary Objects and Control Theory, among others (Ralph, 2013). Because these theories look at different aspects and explain different situations they are sometimes used simultaneously in one study (Johnson, Ekstedt & Jacobson, 2012).

3.2.1 The Complexity Theory

The term complexity generally refers to “the state or quality of being complicated” (Oxford Dictionary, 2014, p.1). In software engineering, complexity either in the technical or organizational sense Galin (2004) refers to “the measure of the resource expanded by a system while interacting with a piece of software to perform a given task” (Kearney, Sedlmeyer, Thompson, Gray & Adler, 1986, p.1). Software complexity is one of the foremost determinant factors which contribute to issues relating to late deliveries and increased product maintenance costs (Banker, Datar & Zweig, 1989). These factors make testing difficult and introduce errors (ibid). Complexity is usually caused by the inter-relationship, inter-action and interconnectivity of elements between a system and its environment (Chan, 2001). Software complexity is not only measured based on the source code (ibid) but there are other aspects on which software complexity can be based, such as requirement, cognitive functional complexity or development process (Serena, 2007). Organizationally, complexity also relates to the difficulties encountered when attempting to implement, verify and understand a system design or components.

Therefore, Complexity theory refers to a “set of concepts” which explain a complex phenomenon (Business Dictionary, 2014). Complexity theory is also defined as “a new approach to science that studies how relationships between parts give rise to the collective behaviours of a system and how the system interacts and informs relationships with its environment” (Agile Development, 2011, p.1). Complexity theory measures the diversity in internal and environmental factors which include departments, customers, suppliers and technology (Amagoh, 2008). This theory consists of integrated ideas from various theories such as chaos theory, computer science, general systems theory, information theory and other related fields which deal with natural and artificial systems (ibid).

The significance in this study is that since it aims to understand the relations between various computational phenomena including computational problems, it draws a researcher's attention to the complexity of essential tasks (Goldreich, 2005). This statement interaction is said to be a way of gaining knowledge where the "value of a hard to compute function applied to publicly available information" is considered knowledge (ibid). In this study, it is also useful in sensitizing the researcher to the complex behaviour which emerges from rules and complex systems where complex systems are viewed as networks of interdependent parts interacting according to those rules (Keith, 2006). However, complexity theory is more appropriate, and is widely applied in technical software development projects. Whilst this theory offers a useful point of reference in the process of this investigation, the focus of the current study extends beyond a mere awareness of the issues of complexity – it also seeks descriptions and explanations of inter-subjective processes, motivations and related practice outcomes.

In the light of the adaptive nature of agile methodologies such as Scrum, the complex adaptive systems theory is also reviewed in section 3.2.2.

3.2.2 Complex Adaptive System Theory

The Complex Adaptive System (CAS) is a dynamic network of a large number of interacting adaptive agents (Keith, 2006) who act on and are influenced by the local environment - both internal and external (Forrest & Jones, 1995; Eidelson, 1997) . In CAS micro agents are considered to interact with each other to create a system (Miller & Page, 2007). The primary focus is on the external environment which enables internal adjustments to changing environments including self-organization (Keith, 2006, Meso & Jain, 2006). Unlike the complexity theory that focuses mainly on the technical complexity of a software system, CAS extends beyond technicality into requirement changes, the interaction between the user and the software product as well as to the satisfaction with the product outcome.

The complex adaptive system theory is often used to explore the spaces between simple and strategic behaviour, pair and infinities of agents as well as equilibrium and chaos (Miller & Page, 2007). It provides a direction on the dynamic interplay among these aspects and how agile methods enable such interplay (ibid). The CAS addresses issues relating to changes during the development process, self-organization as well as of rules. It also outlines key aspects which contribute to the success of developing software products, helping to account for the interaction between people, processes and software products being developed as well as to accommodate the changing requirements within the software development life cycle.

The significance of CAS is that it guides the development of best practices to improve software development processes, for improved product quality. The focus of the current study was to understand ways in which agile Scrum teams can apply and maximize software quality assurance measures. Emphasis must be placed on the process itself, including the testing procedures used by Scrum teams (Meso & Jain, 2006). For these purposes, CAS has been applied in a study by Meso and Jain (2006) to investigate how agile or internet-speed development of IS solutions could be used in volatile business environments. The theory (CAS) was successfully used to develop a better understanding of agile software development methods, and to understand internal quality during the development of a software product. The theory is useful in sensitizing developers of the interactive dynamics among team members in agile projects. This is important, because the software engineering process does not end when the product is released to the customer. In this regard, CAS suggests the prioritization of the user interaction and satisfaction with the developed software product.

In the current study therefore, CAS offered the possibility of a similar insight into the significance of multiple relational factors, including relational aspects in Scrum projects. The application of CAS worked well for this study because of its technical focus. In addition, the theory was found useful in sensitizing the researcher to factors to explore during the investigation, without researcher necessarily understanding why certain quality assurance decision are made or omitted. The reason is that CAS is not designed to explain the inter-subjective aspects of quality assurance choices and decisions in Scrum processes.

The theory of boundary objects which is discussed in section 3.2.3, also offers a useful insight on understanding the balance between needs and constraints in the software engineering process.

3.2.3 Theory of Boundary Objects

Boundary objects (organizational or personal) refer to objects which are flexible enough to adapt to local needs and constraints of the several parties employing them (Pries-heje & Pries-heje, 2011), yet robust enough to maintain a common identity across sites (Holtta, 2013). The Theory of Boundary Objects was developed by Susan Leigh Star in the late 1980's to provide "a deeper and better understanding" (Pries-heje & Pries-heje, 2011, p.6) of communication between key project stakeholders.

The theory of boundary objects has been applied to the information behaviour of users, communities and organizations to contextualize how social behaviours relate to information systems (Worrall, n.d.). In relation to the current study, the boundaries could be a lack of knowledge among project stakeholders. For example, developers may lack knowledge on

how the business operates. Whilst managers may have a high level of understanding of the system, the user may lack technical knowledge, meaning that an effective collaboration among all project stakeholders is important. Communication among all project stakeholders such as users, developers, user management, and management is important (Pries-heje and Pries-heje, 2011). In effect, the Scrum methodology emphasizes close collaboration among self-organizing teams, because communication often becomes a problem in distributed teams (Mahanti, 2007; Pressman, 2010; Tray, 2010). The theory helps the stakeholders understand a way to translate, transfer and transform knowledge between teams that are geographically distributed with limited face-to-face communication (Fong, Valerdi & Srinivasan, 2007). It emphasizes open channels between key project stakeholders and a clear interaction between user and the system.

However, though the theory of boundary objects addresses the importance of managing boundaries among project stakeholders (to improve communication), it is not clear how the theory would help Scrum masters to manage and control all aspects of distributed self-organizing teams. With this aspect in mind, the role of the theory of control was also explored (section 3.2.4).

3.2.4 Control Theory

Control theory is defined as the primary theoretical lens used to understand a process of guiding a team to complete a project (Maruping et al., 2009). When the agile methodology is being newly implemented, managers may have little knowledge of managing teams within the agile environment (ibid). Thus, it becomes important to have controls in place to provide teams with self-sufficiency to determine the methods of achieving project objectives and to effectively manage interactions between control modes. Thus, the control theory is usually used in software development to understand and manage teams.

The control theory helps managers to appropriately set standards for team performance and to meet standards (Maruping et al., 2009). The theory encourages a system to be developed in such a way that it is stable, in the sense that changes in the control input do not result in disruptions in the behaviour of the system (Kokar, Baclawski & Eracar, 1999). In the current context, managerial control would relate to coordination and facilitation of sprints and daily Scrums. In this respect, there are three levels of control which include Scrum meetings held as a way of keeping track of progress and the impediments, control which is usually done after each sprint, and the one that concerns the entire project (Koskela & Howell, 2002). Managers have a responsibility to eliminate impediments. Control theory was applied by Maruping, Venkatesh and Agarwal (2009) in a study to explain the impact of agile software

development on software project quality, and in a study to explain the effectiveness of flexible management practices using empirical data from distributed agile projects (ibid).

In relation to the current study, management comprises three roles. The first is the Scrum master who manages the process. Secondly, the product owner who manages the product and the team that manages itself (Pries-heje & Pries-heje, 2011). During this process a product owner facilitates knowledge sharing where boundary is used to facilitate coordination and transparency.

In conclusion, it is evident that there are various important aspects that make up or contribute towards developing a software product that satisfies and meets user needs as well as delivering the product on time. In order to fulfil these quality aspects, various processes occur during the software development life cycle. To address these factors various software engineering theories can be applied as operational frameworks, either simultaneously or separately, depending on the objective, chosen methodology and the environment where the software is developed. For example complexity theory uncovers complex behaviour and the rules which are applied in complex systems as well as the interaction of these behaviours according to rules. In support, complex adaptive system theory addresses underpinning for dynamic interplay among processes, people and a product where the interconnection of this aspect is regarded as the determinant factor towards delivering and meeting customer needs. Although complexity and complex adaptive theories address some of the important software development processes, constraints which arise are neglected. However, the theory of boundary objects addresses communication constraints among team members.

From the above descriptions it is evident that the use of software engineering theories can play a major role in improving product quality and the software process. However, none of the theories could offer a holistic framework and a lens to adequately analyse all aspects of the software development quality assurance in Scrum projects. For this reason, the structuration theory was adopted and used as an analytical framework in this study.

3.3 The Structuration Theory

Structuration theory is defined as the creation and reproduction of social systems based on the analysis of social systems, structure, structuration, power and agency (Giddens, 1984; Ma, 2010). As demonstrated in Table 1, the structure, system, agency and structuration are the key (and inter-dependent) concepts in structuration theory (Rose & Scheepers, 2001; King, 2012).

Table 1: Structuration Theory Concepts

Concept	Definition of concept
Structure	Structure is defined as a set of rules and resources organized as properties of social systems (Giddens, 1989; Rose, 1998; Rose & Hackney, 2003) from which human agents draw on, and which they reproduce as they act. It also refers to structural properties which allow time and space embedded in social systems.
System	System refers to reproduced relations between actors, organized as regular social practices (Jones & Karsten, 2003). It also relates to social institutions of modern society (King, 2012).
Structuration	Structuration refers to conditions governing the continuity of social transformation of structures and the reproduction of system thus reproduces social systems. It is the production and reproduction of social systems introduced by the interaction of rules and resources (Rose, 1998).

In order to elaborate on the ST summary in Table 1, the essence of structuration theory is presented under the core concepts of structure, system and structuration.

3.3.1 Structure

As a set of rules and resources organized as properties of social systems (Giddens, 1989; Rose & Hackney, 2003), structure is an outcome of an action or an object which can be observed in that it enables human actions (Orlikowski, 1991). Representing the application of rules and resources, structure exists only as memory traces, serving as the organic basis of human knowledgeability that is instantiated in action (Rose & Hackney, 2003). In other words, it does not have physical existence, but a product of human action that is manifested through (and is dependent on) the activities of the human agent (Ma, 2010). In this instance rules refer to routines followed by organizations, individual or teams to accomplish goals. Resources on the other hand, relate to interactions, attributes or materials that can be used to control and influence members. They are structured properties of social systems, drawn on and reproduced by knowledgeable agents through interaction (Rose & Hackney, 2003). Structure is regarded as a set of procedures, tasks, hierarchical relations, regularities and processes of interaction that are embedded in human action.

Whilst rules relate to the guiding and limiting properties such as policies, standards and best practices created by and guiding human actions, resources may appear in both authoritative and allocative forms (Stillman & Stoecker, 2005). Authoritative refers to the co-ordination of activities of human agents. Allocative on the other hand refers to the control of material products - tools (ibid). Rules and resources are utilized by agents when they are interacting with each other, technology or users interacting with technology when using it. Resources are hardware (PCs, monitors, servers), and an example of rules is an access control policy. Access control policy is where it is required that a user's access be removed when the user leaves an organization. In this instance a policy (rules) is created and utilized by agents.

In the software development environment, structure is represented by organizational knowledge, policies and standards, and the application of these to the programming and quality control activities. These structural properties are incorporated by agents during and after the technology development. For example, in an IT environment human agents apply rules (through their knowledge) using resources (hardware, software) to accomplish tasks (Orlikowski, 2000).

3.3.1.1 Rules & Resources as Properties of a Socio-Technical System

In the structuration theory (ST) context, rules and resources have a mutual relationship (Jones & Karsten, 2003), in that they exist in the practice, meaning that rules in a socio-technical environment are embedded in, and are a product of human action (ibid). Human action implies that technology is a social construct, and its evolving development is largely dependent on activity rules such as procedures and standards that guide human action. In this context, human (agent) action represents agency, meaning that structure and agency are inter-connected entities that are related and mutually interdependent (Giddens, 1984). Agency refers to the capacity of human actors (agents) to make decisions and to be able to work independently (Rose & Scheepers, 2001, Rose & Hackney, 2003). Human agency relates to a human's capability to make decisions or a choice, to be able to work independently and the "capacity to make a difference" (Rose, 1998). Giddens refer to the agent as a reason behind what is being done (Naidoo, 2009). Agents produce, reproduce and develop social structures which enable them in their actions. Social activities are recreated by actors as they reproduce conditions that make human actions possible.

Structuration theory claims that structure and agency have a recursive mutual interdependence (Rose & Scheepers, 2001). Referred to as the duality of structure, it is where agency and structure (Rose & Hackney, 2002) interact together to reproduce practices or to "influence change in society" (Wolfel, 2005). For example, structure is the "medium and outcome" agency, whilst agency also has a causal effect on structure (Ma, 2010). Agents then draw on the established set of rules as they act, with compliance encouraged (legitimation) or deviance discouraged (sanctioned), thereby serving to reinforce specific patterns of practice. Power therefore is a key component of agency as it determines legitimation or the sanctioning of rules towards the reproduction of structure (ibid).

Rules are therefore represented by best practices, organizational standards, policies and techniques that are used to form, sustain, change, terminate and recreate information behaviour (Rosenbaum, 2010). Resources on the other hand are represented by the social and/or material elements used to carry out objective-based activities in an organizational setting. Agency allows command over objects, processes and people. Agents, for example

top management, (executive managers, board of directors) create, recreate, draw from and are guided by polices in their day-to-day activities. In this respect, managers are knowledgeable (about social life) agents who exercise rules and procedures to make decisions such as giving commands (resources) to lower and middle level managers who also give orders to operational managers. Orders could be directing (or creating policies for) employees on how to manage materialist resources such as printers, servers, computers. For example, most organizations (if not all) have policies (applying rules to resources) for moving laptops inside or outside the building where each employee is required to have an access card for moving their laptop outside the building. Policies and procedures are re-enforced and modified to produce social structures. The concept of duality of structure is implied in this argument, in that there is a mutual relationship between agency and structure. Agency cannot exist without structure but they exist as duality. In essence, social structures are both created by human agency and are also the medium and outcome of its constitution in other words there is an interconnection between institutions and structure enabled by exercising practices (Bouthillier, 2000; Loureiro-koechlin, 2008). For example, in the day-to-day actions, employees draw from policies, best practices and standards and therefore actions produce and reproduce social structures.

For example, in most organizations an employee is allocated an asset (laptop), during this time rules and procedures are applied to a resources (laptop). For instance, identification tag with a unique reference number is stored and linked to an employee in the software system used for managing assets. Some rules such as access rights maybe also be granted (by an authorized agent) and be applied in the asset to restrict an agent from installing other than work related software.

It is important and critical for organizations to have policies in place, in particular, so that they can follow best practices and standards in a sustainable manner. However, a policy does not exist in isolation but it is exercised by agents to keep the organization operational, hence, a close relationship between people and policies are essential. In essence, changes in social structures and the relations between actors take place as a result of human actions which are enabled by the structures. In other words, policies, standards and best practices exist in and through the productive practices and relationships of human actors. For example, managers draw from their experiences in order to make decisions, to guide, protect and restrict employees from doing things that could put an organization at risk, such as an information leak.

Further, in a technology development project agents are people involved in the development process. In this situation, rules can be technology design standards, security standards and material quality standards. Then resources are commands given to people building the

technology and materialistic artefacts, such as hardware and software. In this process, people use their experience and knowledge to interact and apply standards and make decisions to develop a working product. Decisions involve choosing quality material, technology for developing a usable quality product, which could also serve as a resource for other organizations for maintaining and improving their existing business processes. In this respect, there is a mutual relationship between people creating technology, structure and technology. In essence, technology is itself a representation of rules and resources integrated together to form one unit called a system. The system is developed, changed and used by actors practicing social structures (the capability to make decision, give commands and orders). In other words, technology is a medium of human actions as it mediates their activities. For example, when using technology, users are provided with tools and guidelines which show them how to perform tasks. Structure is produced and reproduced through the interaction of agents exercising social practices to create a system.

3.3.2 System

System refers to reproduced relations between actors, organized as regular social practices (Jones & Karsten, 2003). It relates to the continuity of social systems over time and space. Systems are produced by the actions of people creating structures (Bouthillier, 2000). A social system is comprised of social relationships reproduced over time and space where structure plays a role in recursive reproduction of a social system. Social practices are the social construction reproduced over time and space through human practices. Social practices also relate to interactional systems performed by human activities. In essence, social practices can be in a form of creation, control and regulation (Barratt-Pugh, 2007). In a software development environment for example, developers (agents) who interact with coding standards, create and recreate social practices. Social structures can be in form of allocating work to other developers: for example in a code review process, where a senior developer authorizes code (making sure that standards are followed) before checking in into source control.

According to Berends, Boersma and Weggeman (2003) social practices are the recurring actions of individuals (agents) which create and recreate systems and in themselves also create structure. In this instance structure exists as a property of social practices, it makes social practices possible and it is also simultaneously reproduced by practices (Rosenbaum, 2010). The practices are produced by agents' interaction drawing from structures. During interaction between agents and existing social structures, new social structures can be created. A new social structure is represented by a group or organization and the behaviours engaged in order to pursue an organization's goals (Quizlet, 2014). The system is

reproduced by the relationship between actors organized as social practices (Giddens, 1984). A social system is comprised of information systems, among other systems, that use software and other components to manage information (Liong, 2004). The software development in particular involves numerous actors such software engineers, project managers, and users. These individuals interact together exercising rules in the form of best practices, business rules and coding standards to develop a working product that meets their goals.

Actors interact with and draw from social structures to form structuration. Structuration is outlined in 3.3.3.

3.3.3 Structuration

Structuration refers to conditions governing the continuity or the transformation of structures and the reproduction of social systems (Rose & Scheepers, 2001; Rose & Hackney, 2002; Rose & Hackney, 2003; Naidoo, 2009). It is a process where social systems such as societies and organizations are reproduced (Jones & Karsten, 2003) through the use of rules and resources by agents (interacting with structural features) practicing practices in one instance, or being transformed (Bouthillier, 2000; Rose & Scheepers, 2001; Rose & Hackney, 2002; Rose & Hackney, 2003; Naidoo, 2009).

In an IT services organization, continuity relates to the process that focuses on the recovery of IT services delivered to the business. For example businesses use frameworks such as an Information Technology Infrastructure Library (ITIL) as a guide to ensure that an end-to-end business environment continues even if a serious incident occurs (Underwood, Bronson, Porier, Weber & Wyatt, 2013). In this process standards and methodologies are employed. These include performing risk assessment for core business functions and an IT service to identify assets, threads, vulnerabilities and controls (ibid). Further, in a software development project, continuity of structure is attained through the reproduction of best practices, architectural standards and coding standards where developers are guided by the standards when developing a software product. Continuity can be in form of training other developers about the way standards (including software and architectural standards, naming conventions) are followed in an application.

Business processes, needs and objectives change, and these changes require alterations to be made to existing software products and standards. This can become a problem if it requires a huge change, which, if not done, can affect the business negatively. Hence a need for the amendment and or creation (transformation) of a new structure is needed. For example, if the product architecture does not meet business requirements, it is difficult to add new requirements. In this situation a solution would be to change the rules and

procedures, or to move from an old to a new technology. Technology changes day-by-day, better ways of doing things emerge, therefore the development of new product may require changes in standards (architectural). In ST this process represents transformation.

In ST action and structure operate as a duality, simultaneously affecting each other (Gregor, 2002). Systems and structures are produced by human actions while producing and reproducing structures which exist as dual relationship in an ongoing cycle through agents' use of rules and resources (DeSanctis & Poole, 1994).

Structuration theory was invented in 1984 by Anthony Giddens to explain and integrate functionalism and structuralism with an emphasis on social structure and interpretivism (Naidoo, 2009). Functionalism is a sociological perspective which consists of different but related parts with each other serving a particular purpose. In this view social structures and social behaviour in terms of components of society and functions are implied (Chegg, 2014). Software development in particular, consists of agents occupying different roles. They may range from those of a customer, who specifies business requirements, to a lead developer who oversees the development process, guides the development team and enforces standards (including security, coding, performance, quality) by consistently reviewing source code. Depending on the size of a project, there may be a team of testers who do different kinds of testing to ensure that a software product is fit for purpose. However, this role is sometimes assumed by developers themselves.

The goal of structuration theory, in a socio-technical environment, is to understand, describe and explain the nature and the relationship between human social actions and technology (Parker, 2010), as well as the interaction between the knowledgeable and capable social agents (Wolfel, 2005). However, structuration theory may also be used as a mechanism to explore learning in the workplace (ibid). Structuration theory enables a researcher to understand "how human agency creates social structures" (Lyytinen & Ngwenyama, 1992, p.3). The theory may also be used to conceptualize the link between the context and process in society (Pettigrew & Mckechnie, 2001) in that it helps to understand the society, organizational and personal contexts within which self-service technology is embedded.

In this study, ST is used as lens to understand how users interact with technology, as it has already been adopted in numerous technology related studies. Some of the studies, that are related to the current one, where ST has been adopted, are discussed in 3.4 below.

3.4 Relevance of Structuration Theory

Socio-technical phenomena such as Information Systems and other IT aspects including software applications are regarded as rule driven systems. This is particularly so for

enhanced communication between people over space and time, consisting of human or information and technology centred perspectives (Rose & Scheepers, 2001). These are the important aspects which are used by developers in software development, for end-users (actors) (Lyytinen & Ngwenyama, 1992; Loureiro-Koechlin, 2008). In ST, humans are conceptualized as agents practicing rules (quality standards, coding standards) whilst interacting with each other and the business (organization) during the development process (Wangler & Backlund, 2005). Software development is also a social activity where managers, end-user and system developers work together to develop a software product that meets organizational needs, and the technical requirements of the system (Rose & Scheepers, 2001; Ewusi-Mensah, 2003). It is for these reasons that ST is found relevant in IS and other socio-technical projects. In effect, ST has been used as a framework and a lens to view a multitude of complex socio-technical research projects in a technology environment, with great success.

3.4.1 A Successful Use of ST in Similar Studies

The ST has been applied successfully in several studies including IS research, software process improvement, accounting (Dillard & Pullman, 2009) and organizational adoption of information systems (Orlikowski, 1991; Jeffery, 2003; Andrade & Zulia, 2007). The ST has been adopted to study the use of Information and Communication Technology ICT (Stillman & Stoecker, 2005) and in agile software development (Stray, 2010). In software development in particular, the ST has been used as a guide to understand human interaction (social) with technology as the software is built and used (Lyytinen & Ngwenyama, 1992; Faegri, 2011) . According to Veenstra, Melin and Axelsson (2014, p.1), ST has been used “to gain better understanding of the development, implementation and use of information technology, with many scholars in the field of information systems [drawing from] ST”. For example, ST has often proved to be useful in gaining insights from the development and implementation of public sector IT and for identifying factors influencing outcomes, or for explaining (unintended) outcomes (ibid).

In a study by Clear and MacDonell (2011), ST was applied as a framework to analyse technology use in global virtual teams. The purpose of Clear and McDonells’s study was to understand how global virtual teams mediate the use of technology. Structuration theory was used as a lens to understand the context of global software activities as applied to the actions and interactions of global virtual teams. The theory assisted the researcher to break software development activities down into three sub-goals. The first of the three goals was to investigate the role of technology-use in supporting the work of global virtual teams. The second role was to develop and apply a framework for researching technology-use

mediation in global virtual teams and to gain a deeper insight, and to develop frameworks for the guidance of researchers investigating global virtual teams. In this project, ST was found to be a useful lens and was successfully used to give a holistic view of the phenomenon. It helped the researcher to better understand and analyse how and why things happened in real world situations. In the same context, Stray (2010) also applied the ST to compare teamwork in a Norwegian Agile Software Development Project, and to analyse the challenges and benefits of teamwork in global distributed software development. The aim was to develop and understand social factors and the challenges that arise in an agile software project such as Scrum. It is clear from this latter example that ST is an appropriate theory to use in analysing agile software development environments.

In a study by Orlikowski (1991, p.7), ST was also adopted to "reconstruct the concept of technology and to propose a model for investigating the relationship between technology and organization". In this study, technology referred to both software and hardware, where the "duality of technology" concept (as per ST) was used to emphasize that technology and organization are interconnected entities (ibid). Major emphasis was placed on gaining deeper "insights into the limits and opportunities of human choice, technology development and use" (Orlikowski, 1991, p.4) and organizational design. In this project ST assisted the researcher to analyse the inter-relationship between people and technology in an organization. This project highlights the relevance and usefulness of ST in analysing technology projects, and in particular, to clarify that technology projects such as software development are not independent of the goals of organizations. They are developed within, and for, organizations.

In a different but equally useful context, Manuel (1999) used the ST to explore the effectiveness of IS implementation in organizations. According to Manuel (1999), technology as a socially constructed combination between hardware and software, calls for an interaction between technology and human actions in related projects. Therefore, technology is a "product of human actions" (Jones & Karsten, 2003, p.34) in that it is designed, developed and used by humans to improve society. In this instance, structuration theory claims that structural properties do not exist outside of human actions (Ma, 2010). An appropriate example would be in the software development life cycle where human agents (developers) design, develop and make necessary modifications after the product has been shipped to the user. It is clear therefore, that ST is a relevant tool to facilitate the analysis of a socio-technical project involving human and technology interaction – for example, a link between agency and structure (rules and technological resources).

Further, a study conducted by Theng, Pang, Kan, Miano and Tang (2008) also adopted ST as a guide to analyse the usability of e-Learning systems by looking at users' interactions

with the product. The aim of the study was to use structuration theory to study the interactions between people and technology. The study looked at human actions and their effects on institutions, by identifying the positive and negative consequences or risks that may affect the usability of the e-Learning system. According to Theng et al. (1997), the application of structuration theory in the study was mainly to add a meaning and depth to the collected data, and to demonstrate that rich data can be obtained even if it is collected from a small number of subjects. Whilst this study substantiates the appropriateness of ST in analysing human-technology interactions (a user's interactions with the e-Learning system in this instance), it also shows that ST is equally appropriate in analysing data from small research samples.

In closing, the appropriateness of the structuration theory as an analytical tool in a socio-technical project such as the analysis of quality assurance processes in Scrum projects (as explored in this study) is clearly demonstrated in this section. Whilst this study is socio-technical in nature, involving an interaction between human agency and technology, studies by Clear and MacDonell (2011), Orlikowski (1991), Stray (2010), Manuel (1999) and Theng et al. (1997) demonstrate a clear relevance of the ST as analytical lens in similar projects. Similarly, the relevance of ST in analysing a relationship between technologies, as reflected by Orlikowski (1991), substantiates the relevance of ST to the context and focus of this study. As supported by this background, the application of ST in this study is presented in section 3.5.

3.5 The Application of Structuration Theory in this Study

The purpose of structuration theory in this study is to explain and interpret functionalism and structuralism, in this argument the software development life cycle (SDLC) is implied. Part of this process includes the analysis and interpretation of requirements which are gathered from human beings based on their understanding and experience of being part of the business process. During a software development process, rules – standards or procedures are used to ensure that the end product complies with defined standards.

According to Orlikowski (2000) during the software development process, developers build technology for various specific uses. Technology is seen as embodying structures, in that human actions are central in the technology development (ibid). In this respect human actions are associated with embedding structures during technology development and actions are associated with ensuring that structures are adhered to when the technology is in use. Therefore, it is important to understand the interactions between people, technologies and social actions. In essence technology is developed through a social-

political process which results in the rules and resources being embedded within the technology (Orlikowski, 1992).

In the IT environment actions, which are made by following practices embedded in social structures, form a recursive relationship (re-enforcing each other) between the structures and human actions in the software development space (Hamman, 2009). During this process, procedures (rules) are enacted in every activity of engaging with social practices (ibid).

As summarized in Table 2 and discussed in the passages that follow - this section outlines the interpretation and translation of ST in this study.

Table 2: Application of Structuration Theory in this study

Concept	Definition of concept
Structure	Rules: coding standards, naming conventions, quality standards (unit tests), peer reviews, code reviews. Code inspection, code simplicity, daily stand-up meetings, and one task at a time, one to four week/s sprint, direct user involvement, and incremental delivery. Rules relate to techniques and conventions for chair-making that are an inseparable part of a given practice (Cockburn, 2001). Other rules/procedures include daily builds, communication rules which could be in a form of coding standards to restrict developers from their short cuts or unacceptable solutions when developing software system (Sohaib & Khan, 2010). Resources: human (developers) and non-human material such as computers, servers.
System	Reproduced relations between agents (developers, Scrum masters, product owners, customer), organized as regular social practices such as best practices, coding standards and organizational standards. It also relates to social institutions of modern society. These include Scrum roles and responsibilities, Scrum meetings and communication procedures.
Structuration	Using existing standards to maintain, add new features and amending standards. Participating in the code and pair review sessions, train and guide junior developers. Allocating work to resources and authorize work during code integration testing.

(Subsequent to) section 3.3, which introduced the structuration theory (ST), the three concepts of this theory are adapted and translated into the context of this study (as shown in Table 2).

The three key concepts, structure, system and structuration shown in Table 2, are broken down and explained further, with all the related assumptions and implications, in the following section.

3.5.1 Structure in a Scrum Project Environment

In this study, the ST term of structure has helped the researcher to articulate the rules and the underlying processes of applying these to resources – as well as to identify the related implications to the quality assurance phenomenon in Scrum development methodologies.

During the software development process, the developer incorporates standards, techniques and procedures (rules) into technology (resources), with rules obtained from best practices, coding standards, quality standards, software architectural standards and standard operating procedures, which could represent an array of social structures. In the SD environment, the structure can then be used in interpersonal interaction such as code reviews and resources - public display screens (Desanctis & Scott, 1994). According to the ST, structure exists in human knowledge. In other words a social system is produced by engaging in social practices during day-to-day human (agents) activities through social practice.

In an agile environment, an ideal structure (application of rules on resources) could be that no team member (agent) should work more than two tasks simultaneously (rule/s). Further, compliance with the rule that every work produced must be validated continuously through testing would be a standard practice. In this ideal environment, team members will be writing a simple code, starting by using unit tests before and after integration testing. During the development process, the team members doing the development have close communication with each other and work closely with the customer, in order to deliver a working product. In the same process, a customer would serve as an information supplier to the development team, providing deliverables as well as adjusting objectives and priorities.

As elaborated in section 3.5.2, during this process numerous rules are applied, these range from planning to coding and testing rules.

3.5.2 Rules and Resources in an Agile Scrum Development Environment

In general, rules are defined as an “authoritative statement of what to (or not to) do in a specific situation” coordinated by an appropriate individual (Business Dictionary, 2014, p.1). A rule can be in the form of a principle, standards or a policy. In an agile environment, rules can be coding, communication and design standards. Agile development rules can be divided into planning, coding and testing. Planning incorporates rules, such as user stories, where the development team writes the system requirements document in the form of a story. User stories serve as a guide to the development team in that they give a clear product vision and a goal. After successful creation of customer user story, the development team estimates the implementation duration of the story. Under a planning rule, a project is also divided into iterations of about two to four weeks in length. In the iteration planning phase, it is recommended that a team start with a meeting to prioritize the work that will be implemented. In this regard, one of the rules is that only items listed in the release plan should be implemented.

The coding rules entail numerous aspects such as collaboration of team members and an interaction between the customer and the development team. During the implementation process developers are guided by coding standards which force them to write code that is formatted following agreed coding standards, naming conventions, in order to produce consistent and simple readable code. Team leaders are responsible for enforcing standards using code review activities where, they sit with the developer before integrating new code with the source control (Team Foundation Server, Subversion etc.). From a testing or quality assurance perspective, developers write unit tests. To measure the progress of a sprint, the development team conduct stand up meetings every morning. The stand-up meeting is coordinated by the team and product owners. As a rule usually only team members participate in these daily Scrum meetings. In addition to rules, there are numerous agile development practices that are used by agile teams as a guide in developing quality products. Agile development practices focusing mostly on Scrum methodology are elaborated in section 3.5.3.

3.5.3 Practices (Agency) in an Agile Scrum Development

From the practice perspective, agile methodology encourages a close interaction between the customer and developers. This interaction commences in the planning phase where the developer team estimate the effort needed to implement customer stories. In this process, the role of a customer is to decide about the scope and timing of the short release. The release consists of the prioritized sprint which should last for thirty days. A sprint is itself a procedure focused on adapting to the changing environment variables, such as requirements, time, resources, knowledge and technology. Core agile development practices include test-first programming, regular refactoring, continuous integration, pair programming and a codebase sharing among developers (Ropa, 2014).

Further, in a Scrum project, there are practices that are adopted by Scrum teams. Even though Scrum does not prescribe which practices to use (Williams, Brown, Meltzer and Nagappan, 2011), it does give some flexibility to choose appropriate practice, depending on the project. Practices that are often used in Scrum projects include continuous integration, iterative development, Test-Driven Development (TDD) and refactoring. Continuous integration entails a developer integrating a piece of work daily in a source control. The integrated work gets associated with the auto-build where unit tests run automatically to verify and pass the code before assigning work to testers. The purpose of auto-builds and running tests is to ensure quality in that errors are identified early, hence reducing time and costs. The TDD is implied in this argument. Unit tests serve as a validation component that

ensures that each unit of program meets customer expectations, that software works after integrating and that potential defects are identified quickly.

3.5.4 System in a Scrum Project Environment

In ST, System refers to reproduced relations between actors, organized as regular social practices (Jones & Karsten, 2003).

In Scrum project contexts, each product has a product backlog consisting of a prioritized list of requirements to be implemented in the iteration by the product owner (an agent). During the development process, one of the responsibilities of a product owner is to authorize product backlog, communicating the product vision to the development team (consisting of seven to ten cross-functional members) and the customer. To go about implementing the backlog, Scrum breaks down projects into a sprint (a list of items to be implemented) which has duration of one to four week/s. A sprint begins with a meeting. The purpose of the sprint meeting is to determine items (agency) that will be included in the next sprint. The selection, sizing and prioritization of items are done by the Product Owner, the Scrum Master and the Scrum team (agents). The Scrum team then specifies how the items are going to be achieved and these items are put into sprint backlog. To manage the progress of the sprint backlog, the Scrum teams conduct stand-up meetings which are held each morning to identify the impediments, if any, what everyone is busy with and the plans for the day.

During the sprint implementation stage, the developer interacts with coding standards, to create and recreate social practices. Social structures can be in the form of allocating work to other developers, in a code review process, where a senior developer authorizes code (making sure that standards are followed) before checking in to source control and/or creating new software standards.

3.5.5 Structuration in a Scrum Project Environment

Structuration refers to systems and structures which exist in a dual relationship with each other to produce and reproduce each other recursively (Felix & Sedera, 2007). In this study, the structuration term has been applied to breakdown and understands interdependent interactions and related implications between the objectives, rules, resources, practices and the embedded operational culture patterns in the Scrum process. The emphasis is on the duality of structure where developers interact with coding standards to create and recreate social practices. In other words there is a mutual relationship between agency and structure.

In this context agency refers to the capability of a developer to make a decision. In this instance a decision could be deciding on the most effective and efficient way/s of writing

simple clean readable, quality code and applying exception handling to critical areas of the solution. It could also be choosing the best way to improve the performance, usability and adaptability of the code. From the adaptability perspective, one can decide to refactor some solution areas that are more likely to be changed frequently, by implementing a design pattern, for example a strategy pattern (Freeman, Robson, Bates & Siera, 2004, p.21). The strategy pattern “defines a family of algorithms, encapsulates each one of them and makes them interchangeable” (ibid). Strategy lets the algorithm vary independently of the clients that use it. A strategy pattern enables an algorithm’s behaviour to be selected at runtime. It encourages class behaviours, which would otherwise be inherited, to be encapsulated using interfaces. The significance of this process is to allow better decoupling between behaviour and a class that uses the behaviour. This means that behaviour can be changed without breaking the classes that use it. For example classes can switch between behaviours by changing the specific implementation used without changing the code. These behaviours can also be changed at run-time (ibid). During this process the developer is guided by existing standards, but can also amend the standards to continue developing quality products.

Further, in the current study context, structure relates to best practices, coding, quality and solution design standards (architectural). Resources on the other hand are in the form of allocating work to other developers (allocative), in code review process where senior developer authorizes code making sure that standards are followed (authoritative) before checking in into source control and/or creating new software standards. In the study context, the continuity of structure (rules and resources) is through the reproduction of best practices, architectural standards and coding standards where developers are guided by the standards when developing software product.

3.6 Conclusion

The aim of this chapter was to present a theoretical underpinning to this current study, in particular the use of Structuration Theory to explore software quality assurance in a Scrum project. The chapter looked at software development processes among Scrum teams, the introduction, the importance and the use of theories in the software engineering discipline, as well as how theories can be used. Software engineering theories such as the complexity theory, complex adaptive system theory, the theory of boundary objects and control theory were found as theories that could possibly be used in this study. However, although each is useful in understanding some aspects that will be looked at, none of them could offer a holistic framework and a lens to adequately analyse aspects of software development quality

assurance in Scrum projects. Hence structuration theory was adopted and used as an analytical framework in this study.

The chapter pointed out that in a socio-technical environment ST can be used to understand, describe and explain the nature of and the relationship between human social actions and technology, as well as the interaction between the knowledgeable and capable social agents. Therefore, in this study, ST is used as a lens to understand how a user interacts with technology, and also as an aid to understanding the society, organizational and personal contexts within which self-service technology is embedded. The three key structuration theory concepts, structure, system and structuration were presented. The examples of how these concepts represent and can be applied in an IT environment, were also elaborated in section 3.3. This chapter also discussed the relevance of ST and how this theory has been applied successfully in various studies. For example, studies by Clear and MacDonell (2011), Orlikowski (1991), Stray (2010), Manuel (1999) and Theng et al. (1997) demonstrated a clear relevance of the ST as an analytical lens in projects similar to the current one. The application of ST in this study is presented in section 3.5, as summarized in Table 2 and further discussed in sections (3.5.1 to 3.5.3).

The significance of this chapter is that it informs the methodology presented in chapter 4 and further applied in chapter 5 (Findings). The next chapter presents the methodology used in carrying out this study.

CHAPTER FOUR – RESEARCH METHODOLOGY

4. Introduction

The aim of this chapter is to provide a description of the steps followed to address the research question. The research design and the research approach followed, as well as the research methods used in collecting and analysing the data. Following the discussion of the theoretical underpinnings to conceptualize the investigation in the previous chapter, this chapter also provides a discussion of the chosen research approach followed in this study, the research methods used to fulfil the objectives and answer the research questions discussed in chapter one. This chapter is divided into four main sections. The first is the research (approach) design in 4.1 followed by the ontology in section 4.2. A discussion of research methodology which elaborates on the research methods used in this study (including data collection and analysis) is presented in 4.3. The chapter closes with a conclusion in section 4.4.

4.1 Research Design

Research design refers to “a detailed outline of how an investigation takes place”, and “specifies how data is to be collected, what instruments will be employed, how the instruments will be used and the intended means for analysing collected data” (Business Dictionary, 2014, p.1). Research design, is also defined as the strategic planning of a study which defines the research type (descriptive, correlational, experimental, and qualitative (Babbie & Mouton, 2001). Research design is important in that it eliminates the boundary between the design and research (Edelson, 2002) as it answers the what and how questions in the research process (Babbie & Mouton, 2001). In essence, it serves as a guide to the planning of what is to be observed and how these observations are to be analysed. For example, in this study the ‘what question’ relates to the exploration of existing software quality assurance processes used by development teams in Scrum projects, the significance of these processes in ensuring software quality, and challenges facing Scrum SQA projects. The how question in this process refers to the methods and techniques of investigating the SQA status in the Scrum environments. Of particular significance for this study here, will be to unpack aspects of the adoption and implementation of quality assurance practices, as well as adherence to related standards by organizations or development teams in Scrum projects.

In this context, the research design consists of a research approach, which outlines the philosophical parameters, and a methodology to carry out the investigation. The research approach adopted in this study thus, is outlined in Figure 3.

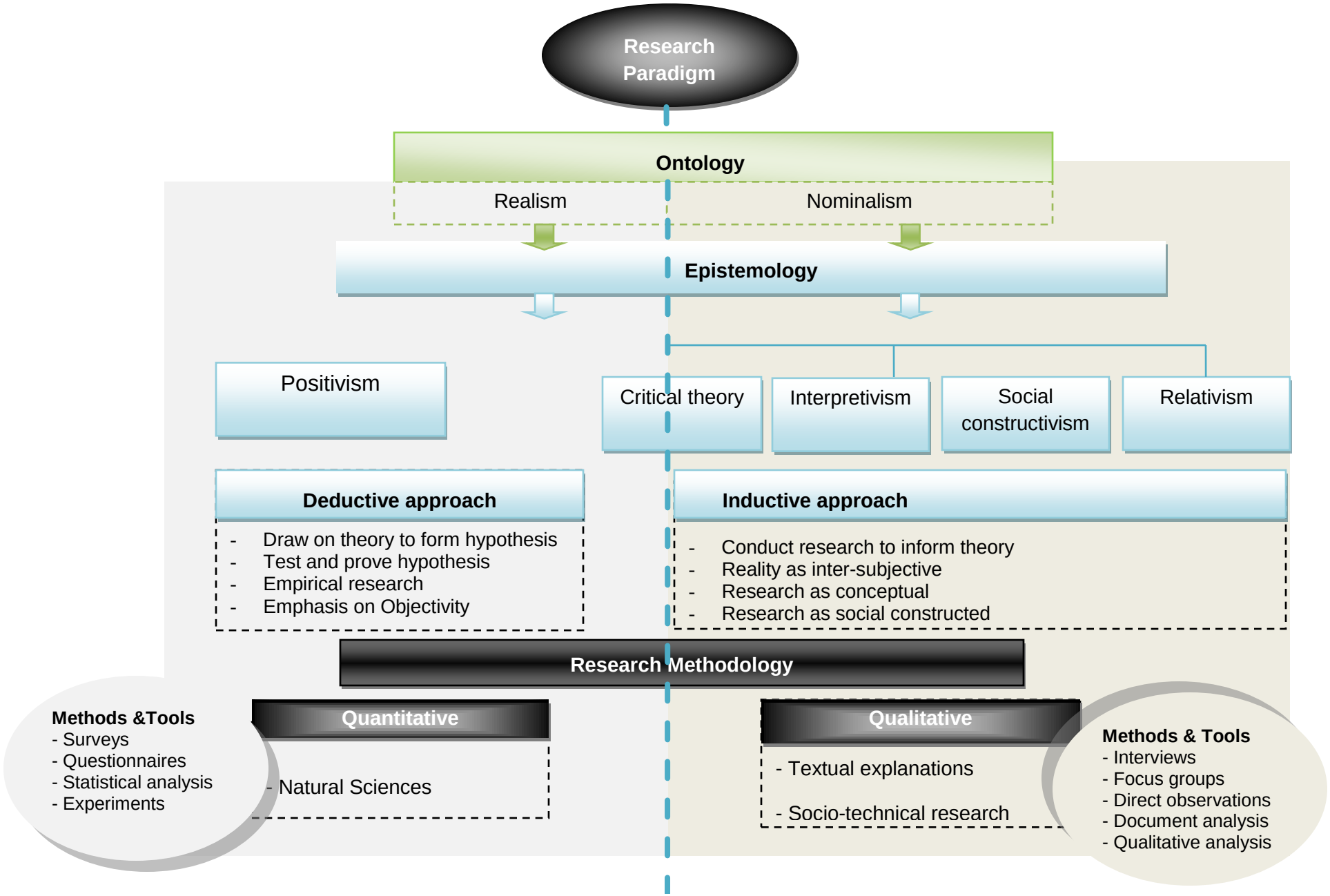

Figure 3: Research Approach Source: Mlitwa, 2014

Figure 3 depicts a research paradigm scenario leading to the research approach that was selected for this study. It consists of philosophical underpinnings including the ontology, epistemology, as well as the research methodology adopted in this study.

4.2 Ontology

Ontology refers to a branch of philosophy that deals with the nature of reality and truth (Krauss & Putra, 2005), in that it is concerned in the general sense, with the nature of what exists and in what form. Ontology can be classified into two kinds of contrasted groups of philosophies: realism (realist or objectivists) and nominalism (normalists or subjectivists). Realist ontology assumes that there is an objective reality, with assumptions of reality mainly consisting of universals rather than particular constructs (ibid). In essence, realism is a philosophy which holds that there is a separate (and objective) reality outside human consciousness, but it awaits to be discovered (Mlitwa, 2011). Realism then is often aligned with natural sciences and empiricist epistemological approaches to knowledge (Uddin & Hamiduzzaman, 2009). At the opposite extreme is nominalism which argues against abstract and universal entities, with nominalists arguing against the existence of invariant structure outside the human consciousness (Hirschheim, 1985). Nominalism therefore, is often aligned with relativist perspectives of reality, and related inter-subjective epistemological epistemologies.

4.2.1 Epistemology

Epistemology is a branch of philosophy concerned with the nature of knowledge and its justification (Krauss & Putra, 2005). It is also defined as the “theory of knowledge (assumptions and beliefs that people have about the nature of knowledge) embedded in the theoretical perspective” (Hesse-Biber & Patricia, 2011). Epistemology deals with the understanding of knowledge and the sources and limits of knowledge (Mertens, 2010). It also addresses issues related to the creation and dissemination of knowledge (relationship between the knower and what is known) in particular areas of inquiry (Bracken, 2014).

Epistemology significantly enables the researcher to develop the research purpose, to design implementation and utilization, which can lead to a rich discussion of issues related to the meaning of objectivity or subjectivity. It teaches its users how to think holistically, helping the researchers to understand and position their study under the appropriate investigation parameters. It questions what knowledge is and how it can be acquired. In this context, knowledge refers to “facts, [beliefs], information and skills acquired through experience,

education and theoretical or practical understanding of subject" (Oxford Dictionary, 2014, p.1). For example, knowledge can be acquired through empiricism (obtained through experiences) or rationalism – which refers to the use of reasoning (Cline, 2014). Epistemology can be classified into different paradigms, including positivism, interpretivism and critical theory, with each approach closely aligned to, and determined by, a particular ontological perspective (Mlitwa, 2011).

4.2.1.1 The Positivist Approach

Positivism entails the testing of hypothesis developed from an existing theory through measurement of observable social realities (Flowers, 2006). In positivism, it is believed that reality is stable and can be observed and described from an objective perspective. The positivist philosophy involves the manipulation of reality with variations in only a single independent variable in order to identify regulations and form a relationship. The researcher draws on theories to construct testable propositions to claims of knowledge (Uddin & Hamiduzzaman, 2009). In positivism, emphasis is placed on values of reasons, truth and the validity of measurements based on factual data gathered through direct observation and experience. The positivist paradigm often draws upon the realist ontological perspectives of reality largely associated with natural sciences where quantitative methods such as surveys, experiments and statistical analysis are common (Flowers, 2006). Therefore this approach is inappropriate for this study since the research problem is about the uncertainties in terms of quality assurance patterns in Scrum teams.

The nature of knowledge dealt with in this study requires the researcher to understand context and the interpretation of results to form a theory hence interpretivism is appropriate for this study.

4.2.1.2 Interpretivism

Interpretivism is a "systematic analysis of social meaningful action through the direct detailed observation of people in natural settings in order to understand and interpret how people create and maintain their social world" (Neuman, 2006, p.76). It relates to methods that adopt the position that human knowledge is a social construction, with a focus on three major social research approaches: social action, socially constructed meaning and value realism (ibid). In this approach emphasis is placed on the ability of individuals to construct meaning. Interpretivism is influenced by Hermeneutics which relates to social science research that originates in literacy studies of textual material, where in-depth exploration reveals a deeper

meaning (Burke, 2007). The main principle is that research must be observed from inside, through the direct experience of the people. Interpretivism acknowledges the constructed and contextual nature of human experience and allows the sharing of realities. In this approach reality is constructed from human knowledge, where the researcher has a direct interaction with participants in a research setting (Thorne, Kirkham & Flynn-magee, 2004). This interaction helps the researcher to get detailed information about the research phenomenon in its subjective context, while collected data can then be interpreted to formulate a theory. In this instance, the role of the researcher is to understand, explain and clarify social reality through the eyes of a variety of participants (Mack, 2010).

The purpose of this study was to understand quality assurance processes adopted by software development teams within the Scrum methodologies. The bulk of this insight requires depth descriptions and explanations of choices and procedures – which requires qualitative rather than quantitative accounts.

The data gathered from different teams operating under the Scrum methodology was interpreted in order to understand experiences shared by Scrum teams. The data was collected using qualitative data collection methods, which are determined by the methodology described in the next section.

4.3 Research Methodology

Research methodology refers to systematic methods, tools and techniques on how the research is carried out (Rajasekar, Philamnothan & Chinnathambi, 2006; Mlitwa, 2011). Research methods are the tools used to conduct research and research techniques refers to the instruments used to conduct the research such as observations, recording and processing data. Research methodology can be either quantitative or qualitative in nature as highlighted below (Nawaz & Malik, 2008; Harwell, 2011).

Quantitative research relates to methods and techniques that measure the quantity or amounts (Rajasekar et al., 2006). It is concerned with numbers and frequencies (Neuman, 2006). Common quantitative methods include experiments (laboratory and field experiments), surveys and questionnaires. The questions asked in these methods are identical for all respondents. The questions can be either closed-ended or fixed and are usually asked in the same order. The significance of these methods in quantitative studies is that it is easy to do a response comparison across participants. However, this technique requires a thorough and rigid understanding of key questions and the range of possible responses (Edelson, 2002). In this

instance rigidity refers to a lack of a formal interactive relationship between the researcher and the participant. Participants are often constrained to select between a given set of responses, meaning that they may not be free to answer in their own words.

However, the aim of the current study was to understand quality assurance processes adopted by software development teams within the Scrum methodologies. This process requires in-depth descriptions of processes, procedures and motivations (thereof), as well as related implications and the underlying impact on final outputs. The bulk of this insight requires in-depth descriptions and explanations concerning choices and procedures. Such in-depth descriptions require qualitative rather that quantitative accounts of this research phenomenon. It is on this basis that qualitative research methods were adopted and applied in this study.

4.3.1 Qualitative Research Techniques

Qualitative research refers to a collection of methods which generate words rather than numbers (Babbie & Mouton, 2001; Edelson, 2002). The intent of a qualitative research methodology is to discover some aspects of social life and describe human experience perspectives by exploring the meaning and the purpose or the reality (Rajasekar et al., 2006; Harwell, 2011). The description and understanding of human behaviours entails a detailed description of participants' actions in terms of beliefs and context. Qualitative research enables textual descriptions of a phenomenon within a given research context (Babbie & Mouton, 2001). This kind of research is appropriate for studying the attitudes and behaviours of participants within natural settings.

Quality assurance decisions, processes, procedures and motivations within Scrum software development teams, as well as impacts and related implications in the case of this study - are directly related to issues of preferences, expectations, assumptions and related behaviours of developers and project principals. These are qualitative phenomena that can be explored through qualitative methodological means. Qualitative methods such as focus groups, interviews and direct observations can be used to collect data under this approach (Barbie & Mouton, 2001).

4.3.1.1 Data Collection

Qualitative data collection methods include participant observation, interviews (including unstructured and semi-structured interviews) and focus groups. Based on the nature of this study, participant observations and interviews are used. A motivation for the use of participant

observation technique is based on the researcher's direct involvement in a number of Scrum teams and related projects in his current working environment, making it a potent technique to collect a maximum amount of data.

Participant observation refers to a process where the researcher actively participates in and at the same time observes the activities of the subjects in their natural setting (Spradley, 1980). According to Kawulich (2005, p.2) participant observation is defined as a "systematic description of events, behaviours and artefacts in the social setting chosen for the study". The researcher describes the existing situation by writing his/her observations while actively conducting informal interviews and writing detailed field notes (ibid). This method is appropriate here, as it is used when researching an inter-subjective problem around qualitative data that requires a contextual analysis and interpretation. It helps the researcher to understand the physical, social, cultural and economic aspects in which study participant live.

The purpose of this study was to understand quality assurance processes adopted by software development teams within the Scrum methodologies. The purpose is also to gain detailed information on experiences during the development process. This requires inside information on development processes, the underlying conditions and motivations as well as first-hand experience on whether and how quality assurance process are applied within the Scrum teams, hence participant and direct observations are employed.

Semi-structured in-depth interviews also help in obtaining a more detailed insight from individual participants. An in-depth interview is an open-ended conversation (question and answer) between the researcher and the participant aimed at exploring, in depth, the experiences and feelings on a particular topic, program or situation, from the perspective of the participant. In-depth interviews are usually used to gather data in research studies where more detailed information about an individual's behaviours, views and experiences are required. In-depth interviews work well for participants who do not feel comfortable talking openly in a group (Kitchenham & Pfleeger, 2002). The nature of this study requires in-depth insight of an individual's perspectives and experiences on the dynamics of quality assurance practices in Scrum projects. Therefore, semi-structured interviews were used to gain explanations relating to quality assurance processes (or lack of them), in Scrum development processes in the teams under observation. Since, the context may differ from company to company, from one team to another and may also involve sensitive information which must not be shared between companies, separate interviews with leaders of participating teams is appropriate.

The actual process of data collection however, largely depends on, and was informed by, a correct sampling process where appropriate decisions are made in order to select the correct types and quantity of data sources. The sampling process followed in the proposed research project is motivated for, and outlined in section 4.3.2.

4.3.2 Sampling

Sampling is the process of selecting target participants that represent a research population, in order to obtain information about the whole research population by examining only a part of it (Babbie & Mouton, 2001; Mlitwa, 2011). In order to simplify the process, the research population can be divided into manageable groups and the study can be conducted on each group. Sampling has two categories: probability and non-probability. Probability sampling refers to a process where the numbers and location of the elements of a research population is known to, and is reachable by a researcher, where all elements of a research population have an equal chance of being selected or excluded from a research sample. Probability sampling can be further divided into three types namely: simple random sampling, Stratified sampling and cluster sampling (Barbie & Mouton, 2001).The current study operates over a broad spectrum where it was impossible to access the entire research population since it is sparsely populated around the world. This makes it impossible to count all the numbers and know the entire target research population. Therefore probability sampling is inappropriate for this study. Thus, non-probability sampling techniques need to be used for this kind of study.

4.3.2.1 Non-Probability Sampling

Non-probability sampling refers to the process where elements of a research population are selected by non-random methods. It is often used when the number of elements in a research population is unknown and their locations are difficult to identify (Kitchenfham & Pfleeger, 2002). The research population in the current study falls under this description. For example, the Scrum software development teams are numerous and too widely spread for their complete number and location to be known with full certainty by the researcher. Without certainty on the number and location of all Scrum development teams, it was impractical for a researcher to randomly select a representative sample from a research population of software development teams operating in Scrum projects.

There are many types of non-probability sampling techniques, with each type determined by the nature and objective of the study, the research problem and the type of a research population

under investigation. The three most common non-probability sampling techniques are quota sampling, snowball and purposive sampling.

Quota sampling is a type of nonprobability sampling which describes the research population characteristics (Babbie & Mouton, 2001). Quota sampling is appropriate for studies where there is no list of the research population being studied. Snowball sampling refers to the process of identifying a research population using a network by starting with few participants in a group (Neuman, 2006). In this approach data is collected from a few participants who can be identified and the identified participants are asked to help in locating other participants (Kitchenham & Pfleeger, 2002). Snowball sampling is appropriate for studies where the research population is difficult to locate and identify (Babbie & Mouton, 2001). For example, doing a study of people who live on the street where it is rare to find a participant in one particular location (ibid). This method is often used for hidden research populations which are hard to access. In contrast, the target research population for this study operates over a broad spectrum, therefore snowball sampling is inappropriate. Instead, purposive sampling is employed.

4.3.2.2 Purposive Sampling

Purposive (or judgmental) sampling is a non-random sampling technique that uses various ways to locate possible cases in a situation where participants are difficult to locate (Neuman, 2006). It is often used when the researcher wants to describe a phenomenon about which little is known, or when it is impossible to gather elements by random selection from a list, into the sample (Mchunu, 2013). In this technique, participants were identified and selected according to their direct relevance to the purpose of the study, using the judgment and the discretion of the researcher (Mlitwa, 2011). The research population in the current study fits this description, which makes purposive sampling the most appropriate technique for the identification and selection of Scrum teams for direct and indirect observation, and of the associated individual participants for interviews.

The application of the purposive sampling technique in this study is outlined in Table 3 and elaborated in detail in the sections that follow.

Table 3: Sample Selection

	Research Question: How can software quality assurance (SQA) processes be understood and applied to maximize the quality of software in Scrum projects?				
Theme of Investigation	**Data Source**	**Tools**	**Unit of Analysis**	**Unit of Observation**	**No. of Participants**
Background, methodology, & theoretical basis.	• Literature	• Read, analyse, write	• Books, articles, papers	• Research methodology books, Software Development (SD) journal articles & conference papers.	
SQA processes in Scrum projects	• Literature • Experts	• Read, analyse, write • Interview	• Books, articles, papers • Academics & Industry experts	• SD journal articles & conference papers. • Academics (2 x from NMMU); and Experienced practitioners in SD: Team lead, Scrum Master (1 x EOHMC; 1 x Old Mutual; 1 x Truworths; 1 x Saratoga; 1 x ScrumSense).	2 academics 7 Practitioners
Significance of SQA in Scrum projects	• Literature • Experts • Practitioners	• Read • Interview • Interview & observations	• Books, articles, papers • Academics & Industry experts • Scrum team leaders & Scrum teams	• SD journal articles & conference papers. • Academics (2 x from NMMU); Experienced practitioners in SD: Team lead, Scrum Master (1 x EOHMC; 1 x Old Mutual; 1 x Truworths; 1 x Saratoga; 1 x ScrumSense).	
SQA awareness among Scrum teams SQA practices in Scrum projects Projects success rate in Scrum projects Explanations to current success/failure rate	• Practitioners	• Interview & observations	• Scrum team leaders & Scrum teams	• Team lead, Scrum master (1 x EOHMC; 1 x Old Mutual; 1 x Truworths; 1 x Saratoga; 1 x ScrumSense).	

In an attempt to realize the objectives of this investigation, the research question was split into five main issues/themes of investigation.

The first of these themes (issues of investigation) pertains to ascertaining the exact software quality assurance (SQA) processes that exist in Scrum software development methods. Insight on this theme is considered a basis upon which the rest of the questions can be built. Software development (SD) books, journals and conference papers, as well as insight from experienced academics and industry experts were used as data sources to answer questions under this theme.

For this theme in particular, two participants were chosen: two academics, one of whom is an experienced researcher and a lecturer, with many years of practical industry experience. They were selected from the departments of information technology and computer science at the Nelson Mandela Metropolitan University (NMMU). On the basis of their direct academic expertise and practical experience in agile methods and related software issues, they were considered to be the most relevant sources of data for this theme. The university department was chosen purely on the basis of convenience to the researcher. The researcher is familiar with the institution and using academics in these two departments simplified access to data sources.

For obtaining answers for all themes, ten industry practitioners were chosen from the participating companies. The participating companies are EOH Microsoft Coastal (EOHMC), Old Mutual South Africa, ScrumSense, Saratoga and Truworths.

The group of industry practitioners consisted of a Scrum-team, team leader and a Scrum-master, from each of the Scrum teams, across the five participating software development companies in South Africa. Participating companies in this respect are EOH Microsoft Coastal (EOHMC), Old Mutual South Africa, ScrumSense, Saratoga and Truworths. They were selected on the basis of their common use of Scrum methods in their projects as well as their easy accessibility to the researcher. Each team leader and a Scrum master are directly involved with all aspects of Scrum processes. This includes planning and supervising projects, implementing SQA standards and ensuring that a quality product is delivered to the customer.

The second theme/issue of investigation, the significance of SQA processes, focuses on assessing the importance of SQA, for example, whether it is considered to be a really necessary part of the Scrum software development processes.

Building on this theme, the third issue of investigation – SQA awareness, is directed at exploring whether (and to what extent) SQA is known among Scrum teams.

The SQA practices in the fourth theme focusses on understanding the patterns which emerge about how SQA is (or is not) being implemented in Scrum teams.

Irrespective of the status of SQA implementation, the fifth theme is designed to explore the success rate of Scrum projects, with inferences related to quality assurance practices.

Questions pertaining to explanations of the status quo are captured under the sixth theme – explanations referring to the success/ failure rate of Scrum projects in each of the five teams. The sources of data to the rest of the five themes are the team-leaders and Scrum-masters from each of the Scrum teams at EOH Microsoft Coastal (EOHMC), ScrumSense, Old Mutual South Africa, Saratoga and Truworths companies in Cape Town, selected for similar reasons as outlined under the first theme.

Qualitative analytical methods were then used to analyse the data obtained from these themes of focus.

4.3.3 Data Analysis

Data analysis refers to a systematic process of inspecting, cleaning and transforming as well as of modelling raw data gathered from different data sources with intent to discover useful information (Neuman, 2006). In this process, raw data is broken down into manageable chunks and converted into new knowledge (ibid). Data analysis incorporates the organizing, categorizing and the summarizing of data. While summarizing data, themes and patterns are identified with intent of transforming collected data into credible evidence (Kawulich, 2004). The purpose of data analysis is to interpret and convert raw data into a meaningful data that describes the phenomenon or participants' view (ibid). When collecting qualitative data through observations or interviews, the researcher identifies problems and concepts that might help to understand the situation under investigation and documents them in research notes with the purpose of identifying important statements (Morril et al., 2000, p.521).

As already mentioned there are two methods of data analysis: quantitative and qualitative. Quantitative methods include frequency distributions, statistical analysis, descriptive analysis, correlations and linear regression (GAO, 1992; Taylor-powell, 2003). However, since the current study is working largely with qualitative data, qualitative analytical techniques were used in this research.

Methods used under qualitative analysis include ethnographic analysis, narrative analysis, phenomenological analysis, interpretive analysis, discourse analysis, grounded theory analysis, constant comparative and content analysis (Morrill et al., 2000, p.521; Mlitwa, 2011; Onwuegbuzie, Leech & Collins, 2012).

4.3.3.1 Content Analysis

Content analysis is a systematic qualitative data analysis method of studying and analysing communication (Kawulich, 2004) and social life by interpreting words and images from documents, film, art, music and other cultural products or media (Crossman, 2014). In this technique, procedures are used to make valid inferences from text. Inferences incorporate the message sender, the message itself and the audience (ibid). It determines the objective, meaning or effect of any type of communication. Content analysis works best for interpretive research studies that have textual data (Kondracki, Wellman & Amundson, 2003) obtained from field notes, open ended questions, focus group, observations and open ended interview responses (Mlitwa, 2011). It is mostly used for understating social themes, cultural change, changing trends in the theoretical content of different disciplines, verification of authorship, changes in the mass media content and nature of news coverage of social issues or problems (Hsieh & Shannon, 2005).

In essence, the current study deals with large volumes of contextual and subjective qualitative data obtained from direct (but non-participant) observations and open-ended interviews. For this reason, content analysis was considered to be the most appropriate technique for the analysis of the textual qualitative data in this study. However, the concept trustworthiness is critical, not only in the sampling process and the design of questions, but also in the analysis and interpretation of data. In qualitative studies trustworthiness often relates to dependability and credibility of the findings determined by the unbiased researcher's degree of confidentiality (Cameron, 2011). The concept of validity and reliability are key to the phenomenon of trustworthiness in research. In this instance, validity refers to the extent to which the research instrument is appropriate to the measurement process (Mlitwa, 2011). The question that is usually asked in this respect is whether the tool is measuring what it was set out to measure (Babbie & Mouton, 2001). Reliability on the other hand pertains to issues of consistency in the tool and processes of measurement, in research. For example, it is concerned with the questions of whether the use of particular tool can yield similar result in a similar process in a different environment and time. The method of analysis therefore, should also be in line with the research approach and related data collection methods, if the findings are to reflect an accurate

interpretation of a research outcome. Thus, it is important that data analysis methods be used consistently throughout the analytical process (Zhang & Wildemuth, 2005). In effect, Illinois University (2014) argues that when analysing qualitative data, aspects such as stability, reproducibility and accuracy should be considered.

In interpretive studies therefore, content analysis is applied in line with the interpretive tradition of data analysis including the hermeneutics principle with emphasis on unpacking possible interpretations of data that have multiple meanings (Mlitwa, 2011). The process also incorporates theme identification, coding and ultimately, the iterative translation and interpretation of data from these themes. Since this works with interpretive data with multiple possible meanings, the hermeneutics aspect enriches the process in the selected analytical technique.

In the current study coding relates to organizing data into categories and sorting it according to the issue of investigation. Within each code there are sub-categories. For example testing can be divided into types like performance, usability and functional testing, which all fall under one theme (SQA processes).

The first step in this study thus, was to go through the research notes collected from the interviews and direct observations. The text data was broken-down into smaller chunks ready for coding. Coding refers to a process within data analysis which occurs prior to data interpretation, where data is re-organized by assigning a unique code to each theme reflected in the data chunks. A chunk of coded data can be a sentence, a phrase or a word. In the current study these chunks are obtained from the interview transcripts and direct observations' notes (Fereday & Muir-cochrane, 2006). Similar chunks (by theme) are grouped under the same code. This enabled the researcher to examine and compare the frequencies of the codes.

4.4 Conclusion

The aim of this chapter was to present a detailed description of the research approach used in this study, including the philosophical assumptions made, the descriptive and interpretive research design followed. The methodology and techniques appropriate to carry out this study were described and presented diagrammatically in section 4.1 (Figure 3). The aspects pertaining to how this study was carried out started with the research philosophies (Ontology) in 4.2. Following the discussion of philosophical assumptions, interpretivism was identified as an appropriate method for this study, in section 4.2.1. Interpretivism implies an ontological belief that research is constructed and that the contextual nature of human experience and the sharing of realities assumes that knowledge is constructed through the direct detailed observation of people in natural settings.

The description and discussion of the research methodology implemented in this study to collect raw data in the field as well as to analyse the collected data were presented in section 4.3. The research methodology was divided into three sub-sections, the qualitative research in 4.3.1 highlighting the purpose and the objective of qualitative research in line with the nature of the current study. Qualitative research methods (data collection and analysis) such as participant observations and direct interviews were used to collect data. A descriptive content analysis was used to analyse the data collected through face-to-face interviews.

A discussion of data collection methods was presented in (with a detailed discussion on their adequacy in the study). Two categories of sampling, probability and non-probability, were described followed by a discussion of the selected sampling method (purposive sampling) in section 4.3.2.2. The table outlining the research question, which was broken down into five issues of investigation, was presented in Table 3 and content analysis in section 4.3.3.1. The next chapter presents a critical analysis of the findings.

CHAPTER FIVE – FINDINGS

5.1Introduction

The aim of this study was to identify at least five Scrum projects with different teams and operating circumstances, so as to understand common quality assurance processes and practices adopted by software development teams within the Scrum methodologies upon which software quality assurance (SQA) inferences on Scrum can be made.

Following an outline and a discussion of the theoretical underpinnings to contextualize the investigation in chapter three, this chapter presents the findings relating to an investigation into the quality assurance processes and practices used by Scrum teams. The chapter opens with an outline of the analysis process, followed by a presentation and a discussion of data (as well as a conclusion to the chapter). The chapter is structured in five sections. The initial analysis process – encompassing the thematic formulation, data collection and the analysis is outlined in section 5.2.1. This is followed by a descriptive presentation of the findings in section 5.3, and a discussion of the findings in section 5.4. The summary and a conclusion to the chapter are presented in section 5.5.

5.2 Data Collection and Analysis

This section outlines the process followed and gives details of the interviewees that actually participated in the data collection, in section 5.2.1, and an outline of the analysis process in section 5.2.2.

5.2.1 The Data Collection Process

In line with the research objective, insight was drawn from the structuration theory (ST), literature and research methodology to break down a research question into several points of discussion during the interview process.

In essence then, the main research question: “How can SQA processes be understood and applied to maximize the quality of software in Scrum projects” was sub-divided into five major themes. The five issues of investigation became the core of the interview process and discussion with the respondents (as shown in Table 4 below).

Table 4: Number of Respondents

Organization	Title/Role	No of Participants	
		Selected	Responded
EOHMC	1 x SM*, 1 x TM**	2	1
Old Mutual	1 x SM*, 1 x TM**	2	1
Truworths	1 x SM*, 3 x TM**	4	3
Saratoga	1 x SM*, 1 x TM**	2	1
ScrumSense	1 x SM*, 1 x TM**	2	1
NMMU	2 x Academic experts	2	2
	Total no of Responds	**14**	**9**

Scrum Master (SM)*
Team Member (TM)**

As outlined in Table 4, only nine (out of the initial 14) respondents selected from the sample were available, and participated in research interviews. In other words, 5 participants who had initially agreed to participate did not make it to the interview. Four were Scrum team leaders from each of the four organizations: EOHMC, Old Mutual, Saratoga and ScrumSence, Likewise, in the case of Truworths, the Scrum master was unable to honour the interview appointment.
In summary two academic experts from the Nelson Mandela Metropolitan University (NMMU), a Scrum master from each of the four organizations and a team leader from one institution, as well as one quality assurance manager and a software architect were interviewed. As shown in sections 5.2.2 and 5.3; data collected from the interviews and direct observation by the researcher were then analysed, interpreted and critiqued.

5.2.2 Structuration Theory & Data Analysis

The first step in the processing of interview data was to transcribe the audio-record. In a true content analysis fashion thereafter, the key concepts of the structuration theory (ST), which is the structure, system and structuration were used as the major themes upon which the initial analytical process was based. Table 5 overleaf, outlines the three concepts and the emergent sub-themes that captured the status of information emanating from the findings. Firstly, the concept of structure is represented by rules, resources, and ultimately, the application of such rules on resources – to push the perpetuation of normative practices, or for a transformation of such practices.

Table 5: Structuration Theory Key Concepts

Concept	Concepts applied to the study
Structure: set of **rules** and **resources** organized as properties of social systems (Giddens, 1989; Rose, 1998; Rose & Hackney, 2003) from which human **agents** draw on (Yates & Giddens, 1997). Structure and agency interact together to reproduce practices or to influence change in software development environment (Wolfel, 2005). **Agency** relates to human's capability to make decisions or choice, being able to work independently. For example Power to promote compliance or sanction non-compliance of best practices.	**Rules (229) –** Practices & Conventions, Standards, Techniques, Procedures. Thematical inferences in the findings: • *Practices **(163)*** - (1) Sprint planning session (AP – R2)x1, (NP – R1)x7, (JP – R2)x1; (2) Up-front requirement gathering (GT – R6)x1, (AP – R2)x1, (3) Product backlog (NP – R1)x1; GT(OM) – R6)x1; (4) Up-front design (AR – R2)x4, (A – R3.1)x3; (5) Sprint implementation review (GT – R5)x1; (6) Active user involvement (AP – R1)x1, (NP – R6)x3, (LF – R6)x6; (AR – R2)x1; (7) Close collaboration (AP – R2)x1, (JP – R1)x6, (GT – R5)x1, (LF – R6)x1; (8) Unit testing (AP – R1)x1, (NP – R1)x3, (JP – R1)x8, (A – R1)x19; (9) Functional testing (AP – R1)x2, (NP – R1)x1; (10) Automated testing (AP – R2)x3, (NP – R1)x3; (GT – R3.2)x7; (11) User Acceptance Testing – UAT (AP – R2)x3, (NP – R2)x5, (A – R1)x11, (GT(OM) – R3)x3, (AR – R45)x1; (12) Regression testing (NP – R1)x11, (GT – R5)x1; (13) Write test cases (GT – R1.2)x1; (14) Code reviews (AP – R1)x3, (NP – R1)x5, (JP – R1)x5, (GT – R5)x3, (GT(OM) – R1)x1, (AR – R3)x3; (15) Peer reviews (NP – R1)x1, (A – R5.1)x1, (GT(OM) – R4)x1, (JP – R1)x1; (16) Incremental development (JP - R1)x1, (GT(OM) – R1)x1, (AR – R2)x1, (A – R3.7.1)x1; (17) Daily stand-up meetings – 15 minutes long (JP – R1)x1, (GT(OM) – R6))x1; (18) Frequent customer feedback (JP – R2)x3, (LF – R6)x5; (19) Team leader approves & sign off requirements (GT(OM) – R4)x1; (21) Scrum retrospective (GT – R5)x1. • *Conventions **(2)*** – (1) Coding style (A – R5.1) x1x1; (2) Naming convention (AR – R5) x1. • *Standards **(26)*** – (1) Shippable product (JP – R2) x5; (2) Time (NP – R3) x18, (AP – R1)x2; (3) Microsoft architecture standards (GT(OM) – R4)x1. • *Techniques **(20)*** – (1) Demo sessions (NP – R4)x4, (A – R4)x1, (GT(OM) – R1)x4; (2) Audit logging (GT(OM) – R1)x1; (3) Prototyping (GT(OM) – R2)x7, (LF – R6.1)x1, (AR – R3)x1; (4) UML diagrams (GT – R6.3)x1 • *Procedures**(19)*** - (1) Create requirement (AP – R1)x1; (2) Write user stories (AP – R1)x2, (NP – R1)x1; (3) Recording technical requirements in a smart sheet (AP – R1)x1; (4) Prioritize requirements (NP – R1)x1; (5) Release planning (MM – R3.1)x1; (6) Size requirements (AP – R1)x1; (7) Sprint cannot changed or aborted (NP – R1)x1; (8) No developer is allowed to work on production (GT – R3)x1; (9) Client approves & sign off requirements (AP – R1)x3, (NP – R6)x1, (MM – R6.1)x1, (GT(OM) – R1)x2; (10) Client signs off change requests (AP – R1)x1; (11) Document requirement (GT – R7)x1. **Resources (182) –** Allocation & use of human & non-human resources, enablers, facilities & tools. Thematical inferences in the findings: *Human Resources **(163)*** - (1) Product owner (JP – R1)x1, (GT(OM) – R6)x6; (2) Project Manager (GT(OM) – R4)x3; (3) Scrum master (GT(OM) – R6)x1; (JP – R3)x1; (4) Business people/analysts (AP – R1)x4, (JP – R1)x2, (5) Developers(JP – R1)x9, (NP – R1)x12, (AP – R1)x6, (GT(OM) – R1)x15; (A – R1)x15; (6) Testers LP – R1)x5, (AP – R2)x1; (A – R1)x5; (7) Customer (GT(OM) – R2)x1; (8) User (NP – R1)x20; (AP – R1)x8; (AR – R2)x8; (GT(OM) – R6)x4; (JP – R1)x2; (LF – R4)x9; (A – R3.1)x7; (9) (NP – R3)x1; (AP – R1)x2; (GT(OM) – R1)x14; (A – R3.7.1)x1 • *Tools **(19)*** – (1) Source control: Team Foundation Server (AP – R1)x7; (2) Subversion (A – R5.1)x2; (3) Software testing software (black box testing (A – R3.1)x1; (4) Robotic testing framework – Jugular (A – R3.2)x2; (5) Smart sheets (AP – R1)x3; (6) Online spreadsheet (AP – R1)x1; (7) Development skills (NP – R4)x1; (8) Scrum boards (GT(OM) – R1)x2.

Concept	Concepts applied to the study
System: Reproduced relations organized as regular social practices.	Pushing existing relations & related practices **(3)**. Thematical inferences in the findings: (1) Scrum master conducts business questions & workshops with users (AP – R2)x1; (2) Team leader mentors developer on following to standards (GT(OM) – R3)x1, (AP – R2)x1; (3) Scrum master presents the end product to the customer (JP – R2)x1.
Structuration: conditions governing the continuity or the transformation of social structures & the reproduction of system (Rose & Hackney, 2003; Naidoo, 2009, Rose & Scheepers, 2001, Rose & Hackney, 2002).	**Determinants** of the continuity of procedures, practices & standards (**reproduction of structure**) **(6)**. Thematical inferences in the findings – to follow procedures & practices reduce & reduce number of defects in production: (1) Product goes to one of our client resources for user acceptance testing (AP – R1)x1; (2) We encourage code reviews (A – R5)x1; (3) Reduce the bug count from the first go through unit testing (GT – R1.2)x1; (4) We write requirements test plans (GT – R6)x1; (5) We start with requirements, build, & document the process (A – R6.2)x1; (AR – R5)x1; (6) We try to keep our releases small (A - R3.7.1)x1 **Determinants** of changes in standards, procedures & practices (**transformation of structure**) **(2)**. Thematical inferences in the findings – to minimize cost, rigid time consuming practices & optimize relevant processes: (1) We don't use formalized requirement or functional specification document (AP – R1)x1; Suspend developer (A – R5.1)

According to data in Table 5, rules in Scrum software quality assurance projects are represented by standards, techniques, practices and conventions. Similarly, resources are represented by the allocation and use of human and non-human resources, enablers, facilities and tools. The aim of identifying the key components of structure was to ascertain the significance that respondents attached to the quality determining rules in Scrum projects at the outset. This was done by dividing the rules and resources into various sub-themes, under which the emergent thematic patterns in the data could be placed. In this way, the number of times each theme appears could be counted to gauge the emergent level of significance. The identified themes and the number of times each theme emerged from data transcripts are presented under the findings in section 5.3.

The system is an equally significant concept in the ST. As shown in Table 5 the system concept is represented by the patterns of relations that through normative practices are reproduced as regular social practices. In the context of software quality assurance processes in Scrum projects, such relations that are reproduced as regular social practices emerged to be those that are pushing existing relations and related practices. The objective of this exercise was to identify power relations in inter-stakeholder relations that in turn, inform certain quality assurance practices (or lack of them) in Scrum development projects.

Finally, the concept of structuration in Table 5 outlines conditions that determine the continuity, on the one hand, and those that determine change(s) (transformation) in social structures and system/s, on the other. In the context of Scrum software quality assurance processes, determinants of the continuity of social structures were identified as those that follow procedures and practices in order to reduce the number of defects in production. In the same light, the determinants of changes or of a transformation of structures were identified as those which minimize costs and rigid consuming practices in order to optimize the relevant processes. The objective of this identification was to establish the aspects that inform a specific culture of quality assurance practices, and those that inform the potential for changes in conditions, and ultimately, in such practices.

With the understanding of the emergent patterns of the status quo in the initial analysis, the insight from the findings in Table 5 is described in detail in section 5.3. The research question was then broken down into five issues of investigation (themes) that were then used to address the objectives of this study.

5.3 The Descriptive Presentation of Findings

This section presents in depth descriptions of data from the three structuration theory (ST) themes in section 5.3.1, and findings on the main research question as represented by the five issues of investigation in section 5.3.2.

5.3.1 Structuration Theory Thematic Data

As shown in Table 5, Structuration Theory (ST) themes of structure, system and structuration were used to analyse data and to reflect the contextual background of the quality assurance process in Scrum projects.

As argued by Giddens (1984), structure consists of the interaction between rules and resources – which are organized as properties of social systems. In the Scrum Software development context, rules have emerged to incorporate conventions and practices, standards, procedures and techniques. In this respect, the findings (Table 5) suggest that rules are an important part (if not a determining aspect) of the Scrum process. For example, themes that represent rules were mentioned 227 times by all participants. However, the level of significance attached to rules varies between themes, with the flexible form of rules afforded a higher level of priority than that of the most rigid and binding categories (or themes) of rules.

In this instance, Practices and Conventions themes were cited by all participants, with 165 citations across all 7 interview transcripts. Practices and conventions consist of 20 items: Unit testing, User acceptance testing (UAT), Code reviews, Automated testing, Regression testing, Active user involvement, Sprint planning sessions, Close collaboration, Frequent customer feedback, Up-front design, Peer reviews, Incremental development, Functional testing, Up-front requirement Gathering, Daily stand-up meetings, Sprint implementation review, Team leader approves and signs off requirements, Scrum retrospective, Coding style and Naming conventions.

A breakdown of the details concerned with practices and conventions, as cited by participants, is presented in section 5.3.1.1 in the following passage

5.3.1.1 Rules as Practices and Conventions in Scrum Development Environment

As a rules driven environment, software development requires frequent interactive communication among project stakeholders. It is an environment where human (agents) are

guided by rules when developing a software product. The term practice refers to a method or process used in a particular field or profession (Business Dictionary, 2014). In this respect, agile Scrum methodology is described as a practice and procedure driven methodology that encourages and creates "*…an opportunity for the right communication channels and the right people to communicate with one another*" (JP – R1). Whilst communication, methodologies and procedures are mentioned in the same lines with practices, practices seem to be the nucleus round which the meanings of such activities revolve. In effect, practices seem to be emphasized more than standards in the Scrum process. For example, phrases related to practices appeared 163 times in the transcripts, with the testing activity appearing 31 times across the transcript. Unit testing is an agile practice where developers write units (prior to the functional code) of code to test each function's functionality in order to minimise errors that might occur during and after the development process. As one of the most commonly used practices, unit testing serves to validate the smaller unit against user stories or functionality (ibid). As a practice based-process, unit testing emphasizes adherence to verification procedures in the course of the development process. It can be very useful in keeping the code base clean especially if the solution has a million lines of code (JP – R1). An "*if the solution has…*" sentiment however, implies the conditionality of this practice. In other words, compliance is largely situational rather than standards based. In the words of one participant, (for example said), *"…it can also waste a lot of time if you are testing the wrong stuff*", especially when the code base is still small (ibid).

When applied, unit testing helps "*…as you write the code because you get the payback immediately*" (A – R3.6). Sentiments are that it is an important practice, and that whenever it is done, it needs to be done correctly, because if done thoroughly at the right time *"…and at the right level of granularity*" (ibid), it can also help to "…reduce the bug count from the first go" (GT – R1.2). The usefulness of the practice is clearly attached to the benefits it offers as a practice when applied at the time of need, rather than rigid conformity across all development circumstances. The trend, as a result, is that not all Scrum teams fully adopt and comply with this practice. The reason, according some of the teams, was that "*…it can also waste a lot of time if you are testing the wrong stuff*" (JP – R1).

Unit testing however, was not the only valued practice in Scrum processes. User Acceptance Testing (UAT) also seemed to be a significant practice with the second highest number of citations, 23, across the interview transcripts. User Acceptance Testing (UAT) is a formal testing process conducted by users (or customers) prior to the delivery of the product. This testing by the customer looks at their needs, the defined requirements and business processes to

determine whether the development software system meets their acceptance criteria. In describing this practice, for example, one participant said that after functional testing which is conducted by the QA, the product *"…goes to one of our client resources for user acceptance testing*" (AP – R1), then if the user is satisfied, *"…we move to pre-production and then to production*" (ibid). The significance of the UAT process is that it *"…gives a user an opportunity to break and drive and do testing to see if it does what they want*" (NP – R6). Therefore, UAT together with other processes such as Code Reviews – is acknowledged as an important part of the Scrum process.

Code review is a commonly used software development best practice where development team members review each other's code, going through line by line to ensure conformance to standards and procedures before the code can be checked in into a repository. It is one of the most cost-effective early bug detection software development practices. It also helps to ensure that functionality meets quality standards. The value of code reviews was highly rated in the interviews, with most participants agreeing on the purpose and significance. The most significant purpose of this practice according to the five participants for example, was its effectiveness to improve quality by reducing the potential defect count early in the development process (AP – R1; NP – R1; JP – R1; GT – R5; GT(OM) – R1; AR – R3). This process ensures that *"…no developer may check-in without a code review*" (AP, R1). The code review is not only used for code quality purposes. It also serves as a knowledge transfer mechanism for senior team members to mentor junior developers in producing clean and readable code conforming to defined standards (A – R5.1) and quality work (JP – R1). Instead of seeing this as a fixed and generally applicable standard practice however, participants describe the practice as contextual, differing from team to team. In one of the Scrum development teams for example, one Scrum master clearly stated that they *"…do not necessarily have normal, standardized code reviews*" (NP – R1). The same Scrum master mentioned *"…an instance where one of [the] developers made a change*" and the team assumed that "*other developer would actually review the change*" (ibid) but the code never get reviewed and the change was deployed into production. In some teams for example, a code review is done formally where a lead developer sits with a junior developer and they go through the code line by line (AP – R1; GT (OM) – R1).

However, other teams prefer to do it "*…informally*" (A – R5.1) where a team leader reviews an integrated code in subversion. The same approach was also cited in [the cases of] other Scrum quality assurance practices such as automation testing, regression testing, active user involvement, sprint planning meetings, up-front design, incremental development, Scrum

retrospectives, daily-Scrum meetings, sprint implementation review meetings, coding style, naming conventions and close collaboration.

Whilst these practices are clearly valued as significant, this does not imply general applicability in all Scrum projects. Therefore, the extent to which they are sustained as standardized quality assurance practices remains tentative, subjective and unclear. Nevertheless, whether the impact of this approach to quality assurance in Scrum projects affects the quality of Scrum outputs, or not, is discussed and critiqued in more detail under the discussion of findings later in this chapter.

5.3.1.2 Rules as Standards in Scrum Development Environment

Standards (as sub-divided into shippable product, time, and Microsoft architecture standards) were also highly rated, with 26 citations in interview transcripts. In a Scrum environment, an example of a standard would be the use of a shared code library or coding naming convention, which, if followed properly creates a uniform code base which everyone in the team can easily read. Coding standards can be useful in ensuring both system design consistency across the application and the delivery of a better product faster.

The definition, development, adherence and enforcement of standards seem to be significant aspects which play a huge role in the software quality (internal and external) assurance process. In spite of this observation, fewer teams than expected were actually including standards in their 'sprint' sessions. Rather, Scrum teams were putting their focus on developing a shippable product on time (JP – R2), a practice which seems consistent with the Agile Manifesto's principle of delivering "*working software* [as a] *primary measure of progress*" and "*deliver working software frequently from couple of weeks*" (ibid). Time is one the key aspects in measuring the quality of a product, to the extent that customers do not only measure quality based on fitness for purpose, but also on whether a product is delivered on time. That reference to the time factor as a criterion of quality emerged twenty times in interview transcripts, and supports this argument (NP – R3; AP – R1). In line with the time factor, the shippable increment concept also appeared five times in one interview transcripts. The fact that some of the participants see Scrum projects as "*feature driven rather than quality* [driven]", also illustrates this point (NP – R4).

The implications in this respect are that Scrum projects are largely time focused, with an emphasis on the production timeline. It also emerged in the interviews that some of the Scrum teams "*…had* [never] *had a planning session where* [they] *specifically talk about quality*" (NP –

R4). It is clear therefore, that “Scrum does not prescribe quality assurance at all, it does not tell in any way how to do the quality assurance measures or prescribe quality assurance processes for quality assurance measure or anything like that” (JP – R1). Instead, “*it tries to create an environment and it enables it*” (ibid), with emphasis mostly placed on code reviews as the dominant quality assurance exercise. Under this modus operandi, one cannot help but question the adequacy of the quality assurance aspect in Scrum projects in the long term. Although the aims of producing a quality product seem to satisfy customers, to ensure that the product is fit for use and that it meets or exceeds users’ expectations, findings suggest that adaptability and maintainability aspects tend to be overlooked in contemporary Scrum development practices. Some of the teams even cited problems that “*...if the key developer leaves the company they actually have to restart from scratch*" (A – R6.2).

However, it is not clear whether these issues relate to the software architecture (structure), design (database) or the complications in the source code itself. Some of these issues relate to the ways a Scrum methodology or Agile Manisfesto is designed. The aspects of Agile Manifesto review, and product documentation in particular, is discussed further under ‘Descriptive Presentation of Findings’ in section 5.4.

5.3.1.3 Rules as Techniques in Scrum Development Environment

Techniques (as sub-divided into demo sessions, audit logging, prototyping and UML diagrams) were also highly rated, with 20 citations in the interview transcripts. In particular, prototyping and system demonstration sessions were the most acknowledged Scrum techniques with 9 citations each, from 3 participants. In the quality assurance context, according to one Scrum master, prototyping eliminates any misinterpretation of requirements, thereby giving an accurate estimate of what the end product will look like (GT (OM) – R1), which encourages an improved articulation of user expectations in the development of the actual product (LF – R6.1). In effect, “*…to ensure that we* [developers] *give the users what they want we* [developers] *give demo sessions*” (NP – R1). In other words, soon after the analysis phase process for example, system requirements are converted into “*…an interactive model that the client can use*”, thereby helping users to play around with the system – a prototype (GT (OM) – R1). Prototyping is valuable because it enables developers to provide demonstrations of the expected product to the customer with the aim of getting feedback. Similarly, demonstration sessions also let the user “*…see the screens that have been completed*” (GT (OM) – R3).

The significance of both the prototypes and demonstration sessions is that they facilitate a continuous feedback from the client to avoid *"miscommunication, misunderstanding and [requirement] misinterpretation*" (LF – R6). However, prototypes and demonstrations can only be as successful as the diligence in adherence to procedures. Therefore, diligent adherence to procedures is an important aspect of every Scrum process. Such procedures are: the Client approves and signs off requirements, the writing of user stories, creating requirements, prioritizing requirements, sizing requirements, releasing planning, recording technical requirements in a smart sheet, ensuring that Sprinting cannot be aborted, ensuring that no developer is allowed to work on production, ensuring that the client signs off any change request and documents all requirements. Reference to a mixture of these procedures appeared 19 times in different parts of the interview transcripts. Like procedures, practices pertinent to Scrum are significant to this development methodology. For example, the approving and signing off of requirements was the most acknowledged Scrum practice. However, other equally important practices, ranging from user story writing by Scrum masters and developers, the sizing, to the requirement for a client to signoff, were also highly rated in the interviews.

The significance of story writing is that "*…it says what needs to be done and then why*", thereby serving as an input for creating requirements. For example, during the requirement gathering process a Scrum master "*…writes user stories*" (NP – R1) in the form of an end user scenario, and this is used as a base for forming up a sprint (AP – R1). Therefore, without user stories it is difficult to link a requirement to a feature, understand its value to business and why the feature should be created. Once the Scrum completes user stories, the sizing - estimating of development (including testing) time required for completing each user story follows (AP – R1). This helps the Scrum team know whether or not it is possible for a feature to be added in the next sprint (ibid). Obviously, the decision is based on the criticality and the importance of the feature, otherwise "*…if it's urgent we can prioritize it and take into the next sprint"* (NP – R1) but the client decides on the priority of the stories (AP – R1).

Although adherence to procedures and practices is highly rated in Scrum teams, a common opinion is that none of the practices are cast in stone. For example, they are contextually applied, meaning that they remain an option, as determined by the respective nature and urgency of the project. On this point, common arguments were that although Scrum allows requirement changes throughout the development process, it is not always possible. In fact, one Scrum master even said that "*…once we* [they] *are in sprint, sprint can be aborted which is the worst case scenario but we* [they] *try not to*" (NP – R1). With this flexible approach to the

application of rules, practices and procedures in Scrum projects, one could not help but question the reliability of the quality assurance process in Scrum projects. (To this effect) The findings relating to this research question are presented in detail under the critical discussion of findings in section 5.4.

5.3.1.4 Resources

Just as the rules are important in the concept of structure, so are resources. Resources are the tangible objects which are converted into physical outcomes in Scrum projects. This point was even clearer in the findings, with references to one form of a resource or another, emerging in almost every set of statement uttered by respondents during the interviews.

Concepts and themes about resources in interview transcripts can be grouped into human and non-human aspects. The human resources category within the agile software development context had 9 items/roles such as (1) users, (2) developers, (3) clients or customer (5) tester, (6) product owner, (7) Scrum master, (8) business analyst, and, a (9) project manager. The combined total was 163 citations in interview transcripts for all these categories.

Though it is often debatable whether a user has a legitimate claim to the title of one of the organisation's human resources, they play an important role in the quality assurance process. Hence, they are (or should be) involved throughout the development process. Particularly, in the user acceptance testing process, as the end prototype cannot go into production without passing their approval test (NP – R6), a practice which helps to reduce "*…miscommunication, misunderstanding and* [requirements] *misinterpretation* (LF – R6). In effect, it is important to realise that, not having a user involved could result in product usability issues later when the application is in operation (LF – R6.1). Largely, because users know what they want (NP – R6), and *"… in order to make sure the requirements are met, the user sign*[s] *off*" (MM – R6.1) after conducting User Acceptance Testing. Obviously, developers also play a huge role in making sure that user expectations are met (LF – R4).

An equally significant human resource in software development processes is a developer. Developers are responsible for conducting sprint activities such as planning and design and they attend sprint retrospectives. They "*…work against requirements to make sure that they actually build what is required*" (AP – R1). From the Quality Assurance perspective, they collaborate with testers to "*find [and fix] issues quicker*" (JP – R1). They do not only implement the requirement but also participate in regression testing (NP – R3) and other types of testing such as unit testing (AP – R1). From the standards perspective, one Scrum master praised their

developers as being "*…big on commenting the code and to note, as to what is gone in and what has changed"* (ibid).

Testers have also grown in prominence in the Scrum teams human resource lists in recent times, to the extent that teams are relying on them, because they ensure that the product meets users' needs (A – R1). Scrum encourages teams to collaborate with one another, and "*…testers are part of Scrum team*" (JP – R1), working with developers and Scrum masters to find and fix defects early" (ibid). With a leadership role in Scrum projects for example, Scrum masters manage the process and *"…help to keep the team on track*" (JP – R3). Participants in the interview process acknowledged that Scrum masters act like a communication bridge between the product owner and the Scrum team, they remove team impediments and ensure that the team is developing the right product and that sprint goals are achieved.

Despite the value and importance of these role players (product owner, business analyst, and a project manager) in Scrum projects, the significance of these roles was not clear in some teams. In all circumstances however, human resources are always mentioned with reference to the specific tools and non-human resources that they manage and manipulate through specific procedures - to develop and produce the desired products.

Non-human resources and tools therefore, play an equally important role in Scrum projects. For this reason, resources emerged frequently in a number of quality assurance discussions, with a combined total of 19 citations in the interview transcripts. In the development process for example, tools such as source control systems play a significant role. Source code control is a software tool used by Scrum teams to manage and track source code changes during the development process. It keeps all versions of a source code and provides facilities for updating and retrieving module versions (ibid). In this study, the Team Foundation Server (TFS) was the most acknowledged source code tool, with 7 citations. Scrum teams use TFS to create and associate a work item with an implemented requirement when developers check-in (AP – R1). It can also serve as a restriction tool in that developers *"…may not check-in if you don't have the name of the person who reviewed your code*" (ibid). Another tool that works in a similar way to TFS in an open source context is Subversion, which was mentioned twice in the transcripts.

In summary then, it is clear that although certain sets of rules are important in Scrum projects, they are not as rigidly applied as in the traditional methodologies. For example, rules are evident in the form of procedures on how to go about the development function – without necessarily having to stick to the same pattern of rules in all activities. What emerged clearly in this respect is that standards are not blindly followed for the sake of compliance across the board. Instead,

the timely completion of a system that eventually meets stipulations is prioritized. In this respect, the Scrum master and the project manager carry a discretionary but high responsibility overseeing and judging the product development maturity. In practice however, the findings show Scrum masters to be the mere coordinators of the team of developers in a Scrum process. It has continuously emerged from observations by the researcher, and from interview responses of participants that Scrum masters serve as facilitators between developers, the development process, testers and the clients, with the regular reflective team process serving the quality assurance function. The application of rules (practices, standards and procedures) therefore, is flexible. This makes it difficult to pin down the quality assurance system into a definite 'black-box' type of description in Scrum projects. Implications attributable to this process in the rate of turn-over on the one hand, and the quality of outputs on the other hand, are interrogated in more detail under the discussion of findings in section 5.4 later in this chapter.

5.3.1.5 System

From the interview transcripts, the concept of a system is represented by the accepted development procedures and the quality effecting behaviour in Scrum teams. In the findings then, this system is depicted in terms of relations between the Scrum master, the project managers, team leaders and developers. In the Scrum teams observed, this is translated into procedures such as answering questions and conducting business and workshops with users, and the constant feedback available in meetings with Scrum team members (AP – R2; NP – R6). For example, one Scrum master said "*…we have workshops where we ask business questions*" (AP – R2). The Scrum master not only attends to the clients briefing on what they want, they also communicate this and facilitate its production by a development team. In this system, the project manager collaborates with the team leader who mentors developers on following procedures, by ensuring that user expectations, requirements and objectives are met. To illustrate this point for example, one Scrum masters said *"…we work very closely with the client to make sure that everyone is happy with the delivery*" (AP – R3) by making sure that *"…you are actually developing the systems that supports their requirements*" (LF – R6.1). Once the feedback is obtained and the product is ready to be shipped, the Scrum master presents the end product to the customer (JP – R2).

Since the system refers to reproduced relations organized as regular social practices (Giddens, 1984), it is clear in this section, that relations governing Scrum team functions – including reflective meetings under the accepted power relations between team members - serve to reinforce and reproduce the current Scrum development practices. One of the defining practices

in Scrum projects is the code review. These constant verifications seem to be the nucleus of output-driven Scrum development work. For example, one Scrum master perceives "*…code reviews* [as] *good from both quality assurance perspective and also from the training perspective because that's how the [developers] are going to learn*" (AP – R2) as well as a practice where team members *"…hold themselves and each other accountable*" (JP – R3). Developers and Scrum masters see time and the rate of turnover as key considerations, and when these yield high volumes and desired returns, they do not see a need for change.

5.3.1.6 Structuration

Structuration refers to conditions governing the continuity or the transformation of social structures and the reproduction of system. In this study, structuration represents the aspects of reproduction and or transformation of structure – in the form of rules and practices within Scrum projects. A reproduction of structure in this context refers to an attachment to the common rules and practices that are seen as 'the tried and tested' where diversions and change are avoided. Examples include loyalty to specific procedures, standards and practices followed by Scrum teams to develop the code, and to reduce defects in production. These are described as expected practices, procedures, and "…*the way Scrum teams work*". Examples include the involvement of clients in the testing phase to verify and confirm a product's fitness for purpose among others, encouraging code reviews, reducing bug count from the first go through unit testing, writing requirement test plans, starting the Scrum process with requirements, building, and documenting the process, and keeping releases small. Statements to this effect tend to include phrases such as "*things we have to do*" or suggestions that it is "*the correct way to go about development and quality assurance practices*" in Scrum projects. A typical statement from one of the participants in this case was that it is better *"…to start requirements, we have to build the process, we have to document the process so that we can actually do the whole thing*" (A – R6.2) prior to the implementation phase. Some respondents made a strong statement such as "*we have to*" - suggesting that this is the way things are, and have to be done. In this respect, while the developers are developing, the testing team writes *"…test plans for those requirements*" (GT – R6), to verify the requirements and to give feedback to the entire team.

In this study therefore, leaders and members of Scrum teams were very much committed to staying 'agile'. There was no desire to change to rigid standards in their operations, which would give almost negligible room for operational transformation or change.

5.3.2 Summary to the Theoretical Analysis of Findings

To summarize the descriptive presentation of findings then, four key points stand out.

Firstly, the most important rules in Scrum are the use of unit testing, code reviews and User Acceptance Testing (UAT). However, they are not viewed with the same level of significance by Scrum teams. Instead of very strict adherence to these procedures, adherence seems to be contextual, with a significant amount of optional flexibility.

From the structuration theory perspective of structure, resources seem more important than rules in Scrum, with human resources and the skill of developers being the most significant of resources under this agile development method.

On the system aspect of structuration theory, contextual flexibility seems to describe the operational environment in the sense that the same guidelines and methods are applied subjectively in cases that are regarded as deserving. Then, the system is confirmed by all testers.

From the Quality Assurance (QA) perspective, members serve the QA purpose in a Scrum project. In this instance, findings point to the use of continuous meetings to reflect on what has been tested, and to determine whether there are aspects to be changed or improved.

Finally, it appeared in the findings that non-rigidity (discretional flexibility) to modify practices according to the context in Scrum process has in itself become a new norm that is upheld and reproduced. As a result, it is often difficult to articulate the QA process into a definite black-box description in Scrum projects. However, the aim of the study was to understand the dynamics of quality assurance practices adopted by software development teams within the Scrum methodologies.

To reconcile this background from the findings with the research question posed in this study, a more elaborate discussion and critique of the findings is presented in section 5.4 below.

5.4 Discussion of Findings

In section 5.3 the Structuration Theory (ST) and its concepts were used to analyse data in order to improve the understanding of Scrum practices and related QA process.

To address the research objective, this section builds on a descriptive background from the structuration theory analysis, in the previous section, to align the responses of participants with the main research question – “how can software quality assurance (SQA) processes be

understood and applied to maximize the quality of software in Scrum projects?" In this section therefore, data is presented and discussed according to the following five components of the research question – the issues of investigation: Software Quality Assurance (SQA) processes in Scrum projects; Significance of SQA in Scrum projects; SQA awareness among Scrum teams; Projects success rate in Scrum projects; and finally; Explanations of the current success/failure rate in Scrum projects.

5.4.1.1 Software Quality Assurance (SQA) in Scrum Projects

Under this theme, the researcher wanted to understand different SQA processes that are unique or followed in Scrum projects and how they are applied or substituted in a specific Scrum team. Findings from interviews are outlined in Table 6, and discussed in deatil in subsequent sections.

Table 6: Application of Software Quality Assurance (SQA) in Scrum Projects

Question/s	Total No of participants	Responses					
		Code Reviews	CI*	Close Collaboration	TDD**	No Answer	
How should SQA processes be applied in Scrum projects?	14 (100%)	4 (25.5%)	2 (14%)	4 (25.5%)	4 (25.5%)	5 (35.7%)	No of respondents per research item

Continuous Integration (CI)*
Test Driven Development (TDD)**

According to the findings (Table 6), the most commonly used techniques that Scrum teams use are code reviews, test driven development (TDD) and close collaboration. These are the key considerations in their SQA functions and processes. Among the SQA processes, code reviews are one of the most commonly used techniques to improve quality during the development process.

5.4.1.2 Code Reviews

When respondents were asked to indicate how SQA processes should be applied in Scrum projects, they gave various answers with code reviews as the common denominator in many

responses. One respondent for example, said that "*from a quality perspective on development, no developer may check-in without a code review*…" Particularly, because where there are "…*code reviews, the senior can review juniors or intermediates and [they, themselves] can review each other*" (AP – R1). In this instance, quality is mostly assured where a senior would "*sit with juniors and just review the code before they book it in and then we also have the development testing circle before any code gets booked in*" (GT – R1). What emerges strongly in this argument is that code reviews are (or should always be) part of the development process in Scrum projects. Actually, one of the Scrum masters went as far as to describe "*code reviews* [as] *awesome*…" saying that they are a "*very cool way, to first of all get developers to learn how each other codes*" (JP – R1). Whilst reference to "*very cool way*" may have multiple interpretations, it clearly has positive connotations that can be associated with an undisputed form of approval to the review of codes of Scrum processes. Findings from the structuration theory analytical perspective in section 5.3 also indicate that code reviews are a significant aspect of rules in the structure of Scrum. In fact, code reviews are categorized as a practice within the rules theme of structure, where it emerged as the most fundamental of all practices in Scrum. The emergent observation in this section that it is the "*very cool way*" supports this analysis. However, the question remains as to whether this is consistently applied in all practices, or is it just one of those practices that are applied for convenience. Findings in section 5.3 point to the optional approach, with Scrum teams relying more on meetings to decide on any necessary steps in any given case.

However, code reviews are not the only recognized quality assurance practice in Scrum projects. Close collaboration and the test driven development (TDD) techniques were also recognized. Though code reviews help to improve quality and knowledge sharing among developers, close collaboration clearly improves communication among Scrum members, thereby reducing the chances of a misinterpretation of the requirements in the early stages of a Scrum product development.

5.4.1.3 Close Collaboration

The second common feature in Scrum SQA, according to research participants, was close collaboration. Close Collaboration is the process where customers and the development team meet together on a daily basis to better understand what is being developed by the developers. This is to ensure that requirements are correctly understood by project stakeholders and that the final product reflects customer needs. Findings from interview participants on this point suggest that close collaboration is what makes Scrum projects work. Sentiments were that it

creates "…*an opportunity for the right communication channels and the right people to communicate with each other*" (JP – R1). The same respondent emphasized that "*what you need to do is you need to have a right conversation with the right people*" (JP – R1). Therefore, "*you need testers who are speaking to developers and testers who are speaking to business people*" (ibid).

The significance is that reflective interactions help developers to learn from the challenges and mistakes of others. For example "*having a conversation about I am testing this and I tested this yesterday and it didn't work so the developer and tester can sit together and figure out what went wrong and fix it here and there, it does not wait for another* forty five *week iteration or another whole development circle before the developer gets back to it"* (JP – R1). The main advantage according to this participant is that faults can be prevented in time. On the same question, another respondent said "*we have a very collaborative process that we do when the feature gets built, it gets built right the first time as opposed to rebuilding the code to bug fixes*" (AP – R1). What comes out strongly from this remark is a commitment to focus on preventative practice. Clearly, this aspect is a significant part of the Scrum development procedures in this respondent's development practice. This developer (who is also an academic expert) went on to say "*I have always tried to keep the user actively involved in whatever I did … and get continuous feedback from the user to actually make sure you are on track all the time*" (AR – R2). Close collaboration, with active involvement of the key stakeholders in this respect, seems to be among the most significant quality assurance practices in Scrum. A general belief among developers (on the close collaboration aspect) is that "*you actually have to involve them in the whole process – the requirements, the design and the actual development you definitely have to involve your user throughout the process*" (LF – R6).

To address the main research question however, one still need to ask the level of general applicability and enforcement of this practice throughout Scrum projects. Even though close collaboration is afforded, this high level of significance nevertheless, none of the participants could confirm that they follow this practice in all their Scrum projects. It can be concluded therefore, that this is one of the quality assurance practices that are followed often, but there is an option not to.

Another SQA aspect that was cited with a large measure of significance in the literature, and again by Scrum masters was the practice called "*Test Driven Development*".

5.4.1.4 Test Driven Development

Test Driven Development (TDD) refers to a two-level process conducted by the software quality assurance team writing test cases to cover new functionality within short development iteration before developers can start coding. When this practice is followed, the first level entails a customer writing user stories, together with cost estimation by developers. This first level also includes prioritizing requirements and developing customer tests. The second level is where developers start testing the implementation and refactoring of the code to remove redundancy. In both levels, testing seems to be the most important SQA component of the development process.

Whilst test driven development is appraised as an important part of the SQA process in Scrum, the purpose of this research was to understand the level to which SQA processes such as these are being (or not being) followed in Scrum process. It was also to understand motivations for and related implications to the final outputs. In terms of perceptions, unit testing in particular was perceived as an important step towards delivering a quality product. In fact, one Scrum master emphasized that "*unit testing can be very useful*" in first keeping the "*code base clean*" (JP – R1) and to simplify the developers work, as it makes it easy to test one functionality at a time. It is "*very beneficial to unit tests*" (ibid). It helps to "*reduce the bugs*…" (GT – R1.2).

In addition to unit testing, other Scrum teams use other types of testing, such as automated testing, to save time. In this instance one team member said "*we have automated some of our testing so that we don't spend our time in testing but we can actually spend time on development*" (NP – R1). These types of testing are useful in saving maintenance costs as they help to identify defects before the product goes to the customer. However, functional testing is used if there are errors that occur once the product is in use hence functional testing is another equally important means of finding errors that were not detected by the developers.

Therefore, once the developers have completed the implementation including their testing, the quality assurance team conducts functional testing followed by user acceptance. This process commences immediately after developers have completed the customer stories. In this respect one respondent said "*QA team* [conducts] *functional testing* [and] *it goes to one of our client resources for user acceptance testing*" (AP – R1). As a practice of SQA and a measure of product quality, once the development team and quality team are satisfied with the product, the user validates and verifies the product to ensure that it meets specified needs. It is important to confirm that the product conforms to defined requirements at the end of the sprint. To illustrate this one quality assurance manager said "*we've got a very effective black box testing software*

so all our defects popup instantly" (A – R3.1). The significance of black box testing is to ensure that the product works according to user requirements (functional and non-functional). This process can also help to identify faults as early as possible, before the product is released for use.

In addition to black box testing, User Acceptance Testing (UAT) is one of the useful techniques which contribute to ensuring that the team is developing the right product. As one respondent explained, "*we grow our UAT environment, so we try to work towards a position where every single production system has a version in UAT*" (A – R3.4).

In relation to the common software quality assurance processes, only two participants mentioned continuous integration, one respondent said that "*there is some tactic limitation because of the web TFS*" which include *"continuous integration and continuous builds*" (AP – R2). In agreement with the previous respondent, another participant said that they run "*builds every time anyone checks-in anything into subversion or source code repository so that's continuous, we have continuous build*" (A – R2) and "*…have continuous builds from checked out from subversion we set up our builds*" (GT – R3.3). Unlike the situation in the fixed functions of the waterfall methods, the focus in Scrum projects is clearly that of continuous builds and checks, to correct as one develops.

Nevertheless, despite the cited significance of the test-driven processes, Scrum masters feel that the practice is not always necessary, meaning that there are occasions, for example in the case of a small and simpler project where such testing would not add value. On this point, the feeling is that there must be a reason to unit test, it is not necessary to always do it as a cross-context routine, if wasting time and efforts is to be avoided. One respondent pointed out that unit testing can "…*waste a lot of time if you are testing the wrong stuff so you don't want to write unit tests just for the sake of writing unit tests because somebody said it was a good idea, you need to apply your mind*" (JP – R1). So, the context, goal and purpose for testing are all important. What emerged out of this point then, is that there are important SQA procedures that are recognized in Scrum projects, but their implementation is often at the discretion of Scrum teams. The thinking on the unit-testing aspect in particular, is that one "*must unit test at the right time and at the right level of granularity*" (JP – R1).

In terms of the objective of this study however, the question remains as to when are the tests omitted, under what circumstances, whether the pressure that comes with the high volume of work contributes to test omissions, etc., and the extent to which testing omissions contribute to failures and bugs. It is clear from the above passages that there are numerous processes

available for Scrum teams. These processes play a significant role in ensuring that the end product meets the required standard of quality. The significance of each of these processes is explored in detail in section 5.4.2.

5.4.2 Significance of Software Quality Assurance (SQA) in Scrum Projects

Under this theme, the researcher wanted to understand the significance of the SQA process, and the measures used to ensure that the developers comply with practices, in a specific Scrum team. In fact, SQA measures entail delivering the project on time, within the estimated budget, satisfying or exceeding user requirements and conforming to defined standards. The researcher wanted to understand how these conditions are accomplished, how customer satisfaction is measured and how quality standards are enforced in a Scrum project.

Table 7: Significance of Software Quality Assurance (SQA) in Scrum Projects

Question/s	Total No of participants	Responses							
		Improves Quality	Time Efficiency	Defect Reduction	Meeting users' needs	Knowledge Transfer	Cost Efficiency	No Answer	
What is the significance of SQA processes?	14 (100%)	4 (28.5%)	3 (21%)	2 (14%)	2 (14%)	2 (14%)	1 (7%)	3 (21%)	No of respondents per research item

According to the findings in Table 7, the primary objective of SQA processes is to improve product quality by elimininating the number of defects during the development process. According to respondents, the significance of SQA processes is to improve product quality, to identify defects, to write code quickly, to reduce the number of defects, to help to develop a product that meets users' needs, to reduce maintenance costs, to tranfer knowledge among developers through code reviews and finally to simplify the development process.

The perceptions of respondents were sought on the significance of SQA processes during the development process and the maintanance phase, one respondent commented that "*code reviews are good…. for quality assurance perspective*" (AP – R2). As mentioned in the previous passage (5.4.1.2), code reviews are useful in that they help to improve product quality by making sure that the code is tested prior to integrating it into the existing solution. From the Scrum team and testing perspective, in addition to code reviews, participants believe that unit tests also play a huge role in improving quality, making it easy to identify defects quickly and it simplifies coding - it is time efficient (JP – R2; A – R3.6). In this instance, one respondent said unit tests "*help you write the code*…" and decrease development time in that they "…*make development faster*" (A – R3.6) which in the end gives testers enough time to do all kinds of testing prior to releasing the product to users for further testing.

As stated in the definition of quality, delivering a project on time is one of the measures of quality assurance. It is clear from the respondents that, though it is important to incorporate quality assurance processes, delivering the project on time within budget is equally as significant as developing quality and maintainable code. In essence, one participant believes that complying and enforcing SQA processes could reduce development time. One respondent reveals that having processes such as code reviews, Test Driven Development (TDD) and close collaboration, a significant number of defects can be identified and reduced very early and before the product goes into production. This respondent said "*we have incredible low number of bugs. Any bugs that come up in the production system are generally fixed within a day or so*…" (AP – R2).

Further, SQA processes contribute towards knowledge sharing among team members in that they act as a training method to junior developers. In most teams, it is routine practice for senior developers to mentor other team members. In this way, following of best practices and standards is passed on, settles in the memory and becomes a habit. The significance of this process is that it helps improve developers "*…from the training perspective because that's how the guys are going to learn*" (AP – R2). It helps them "…*to learn how each other codes*" and contributes to knowledge transfer and reduces training costs (JP – R1; A – R3.7.1).

However, following processes alone might not help in developing a high quality product. Respondents believe that communication also plays a huge role, particularly, in terms of ensuring compliance to expected measures to ensure the quality of the product.

Table 8: Measures of Quality Software Product

Question/s	Total No of participants	Responses				
		System Demos	UAT*	Customer's Responsibility	No Answer	
How is customer satisfaction measured?	14 (100%)	5 (35.7%)	4 (28.5%)	2 (14%)	5 (35.7%)	No of respondents per research item

User Acceptance Testing (UAT)*
Demonstrations (demos)

Table 8 offers a descriptive summary of the perceptions of the participants on the measures of a quality product. These measures do not only determine the readiness of a product but also provide a feedback from the customer in terms of meeting requirements. Though there are SQA processes used by Scrum teams to improve and maintain product quality, it is also important to interact with users and get some feedback. This process helps to find out whether the product is fit for its purpose. When participants were asked how do they know whether the end product satisfies the customer, they highlighted various QA measures such as system demonstrations, User Acceptance Tests (UAT) and giving the system to the user for testing.

5.4.2.1 System Demonstrations (Demos)

When the participants were asked to indicate how to measure product quality in terms of ensuring that the customer is satisfied, one participant reflected on system demonstrations where a shippable product is presented to the client (customer) and then "…*the client decides before we can go into production*" (AP – R3). The significance of this process is to ensure quality and that the customer is happy with the delivery, by demonstrating the working system against defined requirements. On the same point, another Scrum master added that, the significance of "*demo sessions*" is to ensure that the development team *"give[s] users what they want" (NP – R6).* They ensure that the system is *"…actually meeting the expectations, you are actually developing the system that actually supports their requirements"* (LF – R6.1). Further, from the quality assurance perspective, the demonstration sessions play a dual integral role for both Scrum team and the customer. From the development teams' point of view, it helps the development team to understand the needs and expectations of the customer and the customer

feedback indicates how satisfied the customer is with the product. On the other hand, it helps the customer get a feeling for the product, and to see whether the product serves their needs. On this point, one Scrum masters feel with demos *"…the user can see the screens that have been completed"* and based on what is presented to them, *"…they decide this is what I want" (NP – R6).* Further, the benefit of presenting a *"…shippable increment"* is for project stakeholders to "*see exactly what* [the system does] *and then recommend changes*" (JP – R2). It makes it "…*easier for them to see something then understand oh actually we needed that button in a different place*". For example "…*to go from point A to point B and what we need is to have all that information in one place instead*" (ibid).

System demonstrations are not only beneficial in measuring product functionality but are essential to system usability evaluation. On this point therefore, *"…quality software is actually more about the actual product than the actual code"* (LF – R4), this however, does not mean that internal quality (source code) is not important but it puts an emphasis on the fact that there should be a balance between the internal and the external quality (usability, reliability, maintainability etc.). In other words, according to one respondent, quality software is *"…more about meeting users' expectations, meeting the requirements of the user, supporting the user to make transactions and making sure that the system is usable"* (ibid).

Despite the cited benefits of system demonstrations, Scrum masters and academics feel that this process works best for projects where requirements are fully understood from the beginning of the project. Therefore, if the customer is unsure about the requirements, they are quite likely to be unsatisfied with the system, due to various reasons such as the technology capabilities and unclear requirements. On this point, the same Scrum master (who is also a Scrum coach) said *"…ninety percent of the time what happens is that people do not know how to put it into words accurately what it is that they really need, …it is very hard for them to describe and explain exactly what it is that they need until they have seen a portion"* (JP – R2). It is clear therefore that, though it is vital to meet or exceed stated requirements, quality is measured by various factors. On this point academic experts believe that requirements are not always *"…100% correct"* (AR – R3).

In addition to system demonstrations, as a Scrum practice, when the development team feels that the product is finished, the customer conducts a User Acceptance Test (UAT) to do a final test and if they are satisfied they approve the system.

5.4.2.2 User Acceptance Testing

Following the system demonstration process, the second common measure of quality assurance and customer satisfaction is User Acceptance Testing (UAT) conducted at the end of a sprint. The UAT is the functional requirement verification process (conducted by product owners, product managers and the user or customer) which confirms that the system is fit for purpose. It also determines that users' needs and business processes are correctly implemented and ensure the delivery of a quality project.

In addition to system demonstration sessions, as a quality assurance practice, Scrum teams should conduct UAT where the user is given an opportunity to interact with the system to verify requirements. Under this process, one Scrum master said through UAT *"…we interact with users during planning sessions and during UAT which is closer towards the end of the sprint*" (NP – R6). In accordance with customer feedbacks, UAT sessions help the development team to identify areas that require change and the customer "*get[s] an opportunity to raise*" aspects that need to be changed before moving to production (ibid). In some teams, the UAT process runs concurrently with integration testing, the latter is done by quality assurance teams running test scripts (GT – R4). However, others choose to dedicate this process to a client's resources in their own environment (AP – R1). On this point, it is evident that although all Scrum teams seem to be following UAT practice, there are no strict guidelines on whether or not the clients conduct thorough testing. In other words the question of how UAT is conducted differs from team to team making it difficult to pinpoint a common way of how this practice is applied and used to measure quality assurance.

5.4.2.3 Customer Responsibility

To get a clarification on measures of quality assurance from the customer, the same question was posed to academics. They were asked to give their perceptions on how customer satisfaction should be measured, using information from some of the projects developed by their 3rd year students for companies. One academic said "…*when student do projects for companies, the focus is on the companies to ensure quality of their quality measurements/metrics requirements*" (AR – R5). On the same question, another academic said "…*the School of ICT passes the ownership of their systems 100% to the students, we as a school do not typically get any comebacks from users/companies which the students developed systems for*" (LF – R7). Though there is no formal UAT session for projects developed by students, there is a similar approach which serves the purpose of verifying the requirement. This

process entails students presenting their products to judges. Its aim is to ensure *"…business understanding…, understand the environment they working on and have they been able to translate that understanding to a system*".

In the description of findings (section 5.3), three common Scrum practices which are applied by almost every interviewed Scrum team were identified as significant practices which contribute to developing a quality product. However, findings also show that Scrum teams choose which practices to apply depending on the type of project. Therefore it is not easy to understand which practices contribute towards the development of a high quality product. In addition, if there are challenges related to time, skill and the understanding of the product, then the system ownership to the client and focusing on system understanding might result in maintainability issues when the system is in operation.

5.4.3 Success Rate in Scrum Projects

Over the past few years, the increase in the adoption, and success in the implementation, of agile methods, in particular Scrum methodology, has witnessed improvements.

Under this theme then, the researcher wanted to understand the success rate of adoption and application of Scrum methods in software development teams. From the quality assurance perspective, the success of a project is measured by delivering the project on time, within the estimated budget, by satisfying or exceeding user defined requirements and by conforming to standards. To understand the successful implementation of Scrum methodology, the question "What is the success rate of the project developed under Scrum?" was posed to Scrum masters and team members. The aim was to get clear perceptions on what the measures of a successful project developed under Scrum methodology.

When the participants were asked about the successful implementation of Scrum methodology in their teams, one Scrum master said *"…we are very successful with the Scrum methodology because we are in an environment where there is a lot of turn in terms of requirements"* (AP – R4). On this point, the same participant also added that, Scrum methodology makes it easier for them to manage a project where requirements change often and to set up priorities. It also "…*allows* [us] *to deal with that sort of change in a way that waterfall methodology would not work*", otherwise *"…we would not work very well with other methodologies like waterfall method because there is so much change Scrum works best for us"* especially in locking down a two week sprint (ibid). It is clear from this argument that the emphasis is on delivering a shippable product covering all the agreed requirements within the specified period (two weeks sprint). This

then can affect the quality of a product if there is insufficient testing time. On this point, one Scrum master feels that *"…dates affect the quality"* of a product (JP – R4). In other words, often times *"…it is not the project that fails but it is the agile implementation that fails so agile is about creating better places to work on so that does not require people to revert back to working overtime, killing themselves, trying to finish the project by and in date which more often than not they do successfully at the risk and expense of all quality"* (ibid).

Another Scrum master feels that when considering the three measures of a quality product, their success is based on the delivery of a shippable and functioning product *"…we measure our success rate by having something shippable and fully functional"* (NP – R4). In this instance, one Scrum master believes the Scrum approach *"…has been a saviour for me for success because I have only had success now with these five projects but it would be a struggle if I was not allowed to mix and look at other methodologies like putting agile in because the Scrum board is wonderful*" (GT – R4).

It also emerged from the findings that in spite of the success gained from the gradual increase in the implementation of Scrum methodologies, there are still various challenges facing agile Scrum teams in developing high quality products.

5.4.4 Explanations to Software Quality Assurance (SQA) Challenges in Scrum Implementation

As outlined in Table 9, there are various challenges facing the implementation of Scrum methodologies which play a role in improving the quality of Scrum projects.

Table 9: Challenges to the Success of Scrum Applications

Question/s	Total No of participants	Responses								
		Adaptation	Insufficient Time	Lack of Resources	Developers' Skill Level	Communication Structures	Improper Testing	Task Tracking	No Answer	
What are the challenges facing Scrum SQA?	14 (100%)	5 (35.7)	4 (28.5%)	3 (21%)	3 (21%)	3 (21%)	2 (21%)	1 (21%)		No of respondents per research item

The most cited common challenges to the success of Scrum projects are: difficulties in the way the development team operates, insufficient sprint duration (time), insufficient developers, and not enough skills for developers to deliver on time, communication structures, insufficient testing and inadequate bug tracking.

5.4.4.1 Adaptability (Change Management)

On the aspect of adaptability/change management, one Scrum master highlighted that their team is experiencing issues relating to difficulties in changing the way people work, especially those who have been implementing traditional methodologies, such as waterfall methods, saying "…*change from the people and change management has been hard*" (GT – R6). On the same point another Scrum master said "… *it's difficult to change the way you do things*" (JP – R5). This seems to be the case in many organizations, what makes it more difficult is the lack of openness in the organisation, lack of transparency is also found to be a problem in the successful implementation of these methodologies (ibid).

Further, from the organisational perspective, Scrum teams are also faced with challenges relating to management support, especially during the release management. In the current context, release management refers to a process that "…*guides IT efforts from application development through testing and into production*, [focusing on] *resources on timely delivery of a feature or set of features that the business needs*". This business disruption delays the process resulting in teams not being able to deliver a quality product on time, as release management reduces testing time. On this point one team said "…*we have got a major problem when we want to release,* [management] *want to take a very conservative release approach which takes about four to six weeks roll out and update the product now because of that it means we can only do about four releases a year*" (A – R7.2). This becomes a concern to the development teams especially when it comes to quality assurance because delivery time is also one of the measures of a quality product. It makes it difficult for a team to roll out the system for pilot testing - releasing the system to users for about a week before the system goes into production (ibid). Although Scrum methods seem to be flexible, findings suggest that Scrum teams are still facing a challenge caused by leaning towards traditional methodologies, which means often times that the Scrum process is mixed with waterfall methods. In fact, some teams find that they are unable to work with the Scrum practice of a sprint duration of over five weeks, because teams spend over four weeks developing a product, and thus they are resulting in teams not achieving their goals.

5.4.4.2 Insufficient Development Time

In addition to traditionalists struggling to adapt to new and better ways of doing things (change), due to insufficient time to better understand and follow the Scrum process, insufficient development time is central to the quality assurance aspect. In essence, insufficient time was cited as the second most common challenge to the implementation of Scrum methodology, and this affects product quality. Time, in particular sprint duration, which in most teams is two weeks, affects the way a Scrum team delivers a quality product, in that it is not always easy to finish agreed requirements on time. On this point, one Scrum master confirmed that "*the challenge we have is the time…, we don't always finish on time* [therefore] …*we measure our success rate by having something shippable and fully functional*" (NP – R4). Although it is in the Agile Manifesto to deliver "…*working software over comprehensive documentation*", it must also be remembered that one of the measures of a quality product, time (meeting the delivery time) is also an important aspect.

Apart from teams not having enough time to deliver all the sprint items, the nature of Scrum itself is sometimes a challenge, especially in a corporate environment where one product owner is responsible for more than project. One of the Scrum practices is that teams should attend daily Scrum meeting to maintain the development progress, remove impediments and to ensure that the process is properly followed. However, due to time and the amount of work that Scrum masters deal with, it becomes difficult for one product owner or Scrum master to attend all daily meetings. On this point, one Scrum master highlighted the impact this problem has on the success of a project. Because of the workload in one team, the product owner "…*is never in the office he is in meetings the whole time now we need him in the mornings*" (GT – R6). Due to this problem, the product backlog could not be managed properly, hence projects failed (ibid). In this instance team members believe that time plays a huge role in developing a quality product, but it is important to "… *do just enough things at the right time..., just enough analysis, just enough documentation, …just enough unit testing*" these things can make a huge difference.

From the quality assurance perspective, findings show that Scrum teams are still faced with challenges in terms of ensuring quality. The participants made it clear that it is still difficult to follow the Scrum practice and deliver a product on time, due to various reasons such as accountability (which is currently lacking), the chaotic nature of the Scrum process and insufficient resources.

5.4.4.3 Lack of Resources (Developers)

Although time is cited as the second most common challenge to the success of Scrum projects, there are various factors such as insufficient resources and developers' skills which could affect the delivery of a product. In fact, one Scrum master said, "…*we also got resources' challenges, for example people being on leave*" (NP – R4), the shortage of developers working in a project seems to be another reason why this participant sometimes fails to deliver requirements on time. Despite the shortage of developers and their skill level, even those that are available have little or no experience in Scrum methodology hence it is not easy to finish all the requirements on time. One team member feels that specialisation (a person who has worked with Scrum in previous projects) also plays a huge role in speeding up the process. This team member said "…*I can say and I think the teams that have worked in a more agile manner are generally teams where we end up having more than one person who knows how something works you've shared the effort*" (GT – R6.1). On this point, one Scrum master said "…*people are not used to it*" (GT – R6), because of this reason, "…*team members didn't want to do the Scrum board because they don't understand it*" (ibid). From the quality assurance perspective, findings suggest that apart from insufficient resources, in some teams, Scrum teams are still engaged in the learning process, when it comes to the understanding, application and a proper implementation of Scrum methods. This makes it difficult to be accurate in measuring the success rate of the projects developed under this methodology. Therefore, a question of what quality assurance processes to follow and how to maximize quality remains unclear, since team members are in the process of adapting to Scrum methodology.

The nature of a Scrum process requires transparent communication channels between project stakeholders and a good management support system therefore, complicated communication structures play a negative role in delivering high quality projects.

5.4.4.4 Communications Structures

Participants also mentioned challenges relating to the size of the organisation and to the nature of Scrum methodology which encourages self-organization among team members. In most instances Scrum teams work better in projects where communication is open between team members and the customer. However, this is not always the case in some organizations in fact some companies "…*do not have openness, they do not have transparency, they do not have the trust structures in place*" (JP – R5). These make Scrum implementation difficult resulting in delays in delivering the project on time. The same participant pointed out that Scrum

methodology works best in environments where the communication structure is not complicated but in some instances "… *the communication structures are very complicated*" ending up affecting the project delivery (ibid). Another Scrum master added that in some companies, communication is "…*difficult because there are too many people with input so there is no one owner of the backlog*" (GT – R6). If there are too many hierarchies and too many roles to accommodate in an agile environment, it becomes a challenge to implement Scrum without experiencing time delays (ibid). Further, some teams experience issues with unstable requirements, for example where the customer might change the backlog at any time, but not change the release date. On this point, one Scrum master said "*We have had a lot of issues even when thing has been approved but they turn around and say actually we didn't think this, you have got to do it this way instead or actually can't you just slide this as well*" (AP – R4).

Associating these statements with the main research question of this thesis, it is clear that quality assurance processes in Scrum are flexible, but also problematic. The three quotations suggest that processes are not seamless – mostly because of multiple structures and stakeholders with divergent interests that are involved in the same project. In fact, the last quotation reveals a sad state of affairs, showing that even those phases, where informal methods are used, can lead to frustrations.

5.4.4.5 Insufficient Testing

The testing process was also cited as one of the biggest challenges facing Scrum teams. In some instances, the Scrum process itself is questioned regarding the accuracy and the amount of testing conducted in such a short period of time. Although there were few participants who highlighted testing as one of the factors which affect delivery of a quality product, testing is perceived to be a huge concern to those who are experiencing it. In this instance, one team member (QA manager) said "…*the biggest challenge I believe is to do proper testing*" (GT – R7). On the same point a Scrum master said "*from the time perspective the developers don't have enough time to do all that regression testing*" (NP – R3). As mentioned in the preceding passages, insufficient time to do testing is one of the contributing factors to unsuccessful Scrum projects in terms of quality assurance. However, this does not necessary mean that these projects fail completely, but teams often fail to deliver on time. If they do deliver on time, there is the possibility that developers will be working on fixing defects instead of adding new requirements (NP – R4). This participant also mentioned an instance where one of the developers implemented a change, and because of time, there was an assumption that the code had gone for testing, but it had not. In fact the project was delivered went into production

and they realized later that that code had never been tested and the team were "*lucky that nothing happened*" (ibid).

5.5 Summary and Conclusion

The aim of this chapter was to present and discuss findings to an investigation into the quality assurance processes and practices used by Scrum teams. It started with a description of data collection and data analysis where Structuration Theory (ST) was used to unpack the status quo and the feel of the Scrum process. The chapter was divided into two sections - descriptive presentation of findings in 5.3 and the discussion of findings in 5.4.The ST was further divided into three main themes: Structure, System and Structuration. Structure is divided into rules and resources. In the context of this study, rules are represented by Practices & Conventions, Standards, Techniques and Procedures. These concepts were then used to draw themes from the interview transcripts and the findings are summarized in Table 5.

It emerged from the findings that rules are a very important part of the Scrum process, they contribute in the development of a shippable product. This was supported by the significant number of identified themes from the interview transcripts under this concept. However, not all of them were found important. Unit Test, User Acceptance Testing (UAT) and Code Reviews were viewed as the most important practices in Scrum projects. Although it is important to conform to these rules, some of them are only applied when there is a need. In other words, the developer skill, project type and size have a direct influence on which practices to apply. It also emerged from the findings that rules are not as strictly applied in Scrum as in the traditional methodologies. Rules are flexible in that they can be modified to meet the environment and conditions of the team. This makes it difficult to understand and measure quality assurance in Scrum methodologies. Findings revealed that rules have a direct relationship with resources. In fact, rules and resources were viewed as being equally significant in the Scrum process. Human and non-human resources in particular, were also found to be equally important. From the quality assurance perspective, developers in collaboration with other project stakeholders, like software testers, appeared to be contributing to the development of a quality product.

To answer the question posed in this study, the research question was broken down into five issues of investigation. The data was analysed and discussed according to these five issues of investigations. What emerged from the discussion of findings was that although code reviews appeared to play a significant role in Scrum projects, the question of consistency in the application of this practice remains open. In essence, findings point to the optional approach,

with Scrum teams relying more on meetings than on code reviews to decide on the necessary steps to be taken in any given case. Therefore, it is clear that Scrum teams do not always commit to following this practice in all their Scrum projects. Findings suggest that close collaboration is commonly followed in Scrum practice across the teams and this is what makes a Scrum project works. It can be concluded on this point therefore, that the code review is one of the quality assurance practices that is followed often, but optionally, all the time. For example, some teams choose not to use Scrum methods in small project or in critical projects.

Further, findings show that Scrum practices such as code reviews, not only help in improving product quality but also in sharing knowledge among developers, especially junior developers. Despite the promising improvements and usefulness of Scrum practices, Scrum teams are still faced with challenges. These include insufficient development time (including testing time), lack of resources, low levels of skill - insufficient team members who understand the Scrum process, teams not conforming to the Scrum practice, complicated communication structures in corporate environments and changing the way people work. It emerged from the findings that, due to the flexible nature of Scrum which *"…does not tell in any way how to do the quality assurance measures or prescribe quality assurance processes for quality assurance measure*" (JP – R1), some academic institutions rely on the students for "…*business understanding, understand the environment they working on" and the translation of business and have they been able to translate that understanding to a system*" (AR – R5, LF – R7) and transfer the quality assurance testing function to the client, in order to ensure the quality of their products.

The next chapter concludes the study and suggests recommendations to the findings discussed in this chapter.

CHAPTER SIX – CONCLUSION AND RECOMMENDATIONS

6.1 Introduction

Scrum is one of the fast growing development methods within the range of agile software development frameworks. However, uncertainty with quality assurance practices in Scrum development processes have emerged as a problematic area.

Subsequent to the research process and the findings, this chapter offers a conclusive summary of the process, findings and recommendations of this study. The chapter opens with a summary of the thesis including a summary of the findings, in section 6.2.

The summary of findings reflects on the purpose and the objective/s of the study, the research question/s, as well as whether and how the objective was realized. It builds on a research question to present answers drawn from interview responses, with inferences on the literature and the structuration theory (ST) to inform recommendations. Suggestions on future research are presented in section 6.3, followed by a closing conclusion in section 6.5.

6.2 Summary of the Thesis

The objective of this study was to explore the extent to which software quality assurance processes are understood, and how they are applied to maximize the quality of software in Scrum projects. To this end, five Scrum teams with different operating environments were identified and interviewed.

This section summarizes the work conducted in this study, and puts the findings on a par with the conclusion, as it informs recommendations. The thesis is divided into six chapters, with the first chapter introducing the study. It presents an overview and a background to the research problem, including an account of the significance of technology in our everyday lives. Here, major emphasis is placed on quality assurance in Scrum, as one of the agile software development methodologies, a background which leads to a research problem. A research problem clearly articulates a lack of clarity on quality assurance practices within Scrum process. This observation informed the main research question which is “How can software quality assurance (SQA) processes be understood and applied to maximize the quality of software in Scrum projects?”

The objectives of the study are also presented in chapter one, which is to identify existing Scrum teams and related projects in Cape Town (Western Cape), to observe practices and to interview key members of Scrum teams on their motivations for preferring Scrum processes. The aim of this objective was to understand practices, challenges and related quality assurance processes – so as to improve insight into the discipline and its field of practice.

The researcher explored the literature and presented a detailed review of the status of the quality assurance process under Scrum methodology in chapter two. Emphasis in this chapter was placed on quality assurance in the software development environment, generally, and in agile methods in particular. Chapter two mainly offers insight into how quality is ensured in both Waterfall (looking at each sequential phase) and agile methodologies, such as a Scrum, thus uncovering the whole field of Software Quality Assurance (SQA). Of essence in chapter two then, was that unlike in Scrum methodology which is flexible with emphasis placed on time-efficiency, control, flexibility, adoption and ease of adaptation to changes, in Waterfall methods quality is ensured by using a rigid format. In fact, the literature shows a limited number of studies published in the scientific journals, focusing on quality assurance. Those few studies found on Scrum methods, are reflected in a case by case format which seem contextual and therefore inconclusive. This format made it difficult to pin point quality assurance. As a result it was not clear from these studies, how Scrum teams can maximize quality requirements in Scrum projects. Thus it was difficult for the researcher to get a consensus on Quality Assurance from previous studies that could be used as a point of reference. Hence a practical investigation was conducted to understand how local teams ensure quality in Scrum projects.

Because the field of software development is largely technical and complex with numerous underlying factors, software engineering theories such as the complexity theory, complex adaptive system theory, boundary object theory and the control theory were explored. The aim was to develop a theoretical framework to offer a lens through which a holistic investigation can be based. However, none of the theories (software engineering) could offer a holistic framework to adequately analyse all aspects of the software development quality assurance process applied in Scrum projects. Therefore, Structuration Theory (ST) was adopted and used as an analytical framework to guide the researcher throughout the study. This theory was found to be the most appropriate theory to better understand the effect of product quality in Scrum development. The ST key concepts such as structure, system and structuration helped the researcher to get major themes (and sub-themes) from the interview transcripts. The term structure in particular helped the researcher to articulate the rules and the underlying processes

of applying these to resources – as well as to identify the related implications to the quality assurance phenomenon in Scrum development methodologies. Here resources are represented by the allocation and use of human and non-human resources, enablers, facilities and tools.

Another important concept of ST is system, represented here by the patterns of relations that through normative practices are reproduced as regular social practices. In the context of software quality assurance processes in Scrum projects, such relations are: that are reproduced as regular social practices (pushing existing relations and related practice). Finally, structuration represents conditions that determine the continuity, or those that determine changes (transformation) in social structures and system/s. Structuration offered aspects that informed a specific culture of quality assurance practices, and the potential changes in conditions, and ultimately, in such practices.

The theoretical framework then informed an interpretive approach which was adopted and described in Chapter four. The chapter four presented a detailed overview of the research methods employed to collect and analyse data. This assisted the researcher to understand a philosophy from which the study originates (i.e. nominalism) which then informed the 'epistemological' approach followed.

To answer questions asked in this study, five Scrum teams were selected (based on their direct relevance to the study and the purpose) using purposive sampling. As already mentioned (earlier in this study), there is a limited number of studies published on SQA in Scrum methods which made it inappropriate to randomly select from a list of Scrum teams hence purposive sampling was employed. In line with the research approach (interpretive) adopted in this study, qualitative methods (interviews and direct observations) were used to gather data and provided a descriptive analysis. Interviews were conducted with each of the identified Scrum teams at their premises. The Scrum masters from EOH Microsoft Coastal (EOHMC), Old Mutual South Africa, ScrumSense, Saratoga and Truworths were interviewed to get an understanding of how they apply SQA process in their Scrum projects.

The data collected was analysed and presented in a descriptive format in Chapter five. Data analysis started by transcribing audio recorded interviews into text format. The main themes were drawn using ST concepts, structure, system and structuration. Findings revealed that testers, users and developers among others, play an important role in the quality assurance phase of the development life cycle, and none of these roles work in isolation. For example, developers are the ones who plan, design, develop and even test the product to ensure that customer requirements are met. To meet customer's and team's objectives, developers

collaborate with testers to find and fix bugs quickly. Findings show that testers are as significant as developers in that they also ensure that the product meets or exceeds users' needs.

Further, although human resources seem to play a major role in the quality assurance process, non-human resources were found to be equally significant. For example, according to the findings source control tools, such as Team Foundation Server, for managing and keeping track of work items as well as a restriction tool to other practices, including code reviews, are some of the most common non-human resources used by Scrum teams.

Under the ST concept system, findings show that system represents the relations between a Scrum master, project manager, team leader and the developer, where a Scrum master conducts workshop with users, asking business questions which provide feedback to the Scrum team. The point highlighted in this statement is that power relations which govern Scrum teams reinforce and reproduce current Scrum development practice.

From the structuration perspective, two main concepts such as transformation and reproduction of social structures were presented. In the context of this study, these relate to rules and practices in Scrum projects where reproduction represents the application of common rules and practices, such as adherence to the application of specific procedures and standards when developing a software product, in order to reduce recurring bugs.

The second level of data analysis is discussed in section 5.4 "Discussion of findings", where the findings are presented according to the five issues of investigation: Software Quality Assurance (SQA) processes in Scrum projects; Significance of SQA in Scrum projects; SQA awareness among Scrum teams; Project success rate in Scrum projects; and finally Explanations relating to current success/failure rate in Scrum projects. In this study, the issues of investigation have been used to gather data with the aim of answering the research questions. Among the numerous themes (and sub-themes) that emerge from findings, three commonly applied themes have been identified and discussed under section 5.3 "The Descriptive Presentation of Findings". These are Unit testing, Code Reviews and User Acceptance Testing (UAT).

6.2.1 Software Quality Assurance (SQA) Processes in Scrum Project

From a QA perspective, findings show that there are various useful but flexible processes that are applied by Scrum teams. They are primarily used to ensure that the developed product supports customers' needs (among other benefits). The SQA processes are flexible in the sense that their usefulness and their applicability vary from team to team. For example, some teams choose to apply these processes in certain environments and projects. Some of the most

cited and valued processes/practices include code reviews, close collaboration and test driven development. It appeared in the findings that although some SQA processes seem optional, there are those that are always used. For example, code reviews are one of the most commonly applied agile Scrum practices for both quality assurance and knowledge transfer purposes. However, despite the undisputed significance of code reviews, findings suggest an optional approach where teams are relying more on meetings than on code reviews, to decide any necessary steps in any given case.

Some Scrum practices, in particular close collaboration, seem to be vital in any Scrum project. As a result it was regarded as one of the often applied practices that make a Scrum project work. One of the key significances of close collaboration is that it encourages transparent and clear communication among project stakeholders. In essence, close collaboration is central in Scrum process in that it helps project stakeholders to communicate in the same language, and hence reduces development costs and faults that could affect product quality.

Findings also show that Test Driven Development (TDD), in particular unit testing, which appeared to be applied by almost ,if not all, the Scrum teams, is one of the substantial Scrum practices that ensures that the end product meets the required quality standard. Unit testing does not only help in improving product quality by ensuring that all possible errors are identified and fixed, but it but it also helps in keeping the code base clean, and it simplifies module testing for developers, thus improving development cost and time. Although there is no doubt that unit testing play a major role in quality assurance, findings reveal that, due to various reasons, such insufficient development time, among others, unit testing is not always applied. It is worth noting that some of the interviewed participants feel that although the implementation of unit testing can result in a product of improved quality, it is not always necessary. On this aspect, recommendations are presented in section 6.3 (under recommendations) later in this chapter.

6.2.2 The Significance of Software Quality Assurance Processes

It emerged in the findings that there is a notable significance in each SQA process used in Scrum projects. The significance of each Scrum process varies from one team to another. According to findings, Scrum practices help to improve product quality (by reducing defect count), to reduce development time (if teams conform to processes), to meet user expectations, to improve knowledge transfer and to reduce development costs. From the perspective of quality assurance and meeting users' expectations, the findings suggest that code reviews, in combination with unit testing, ensure that each unit of code (or function) is reviewed, verified

and tested properly prior to integrating the new functionality with the existing source code. This process itself help in reducing the number of defects before the quality assurance team start with different kinds of testing such as integration testing, functional testing and regression testing. Despite the benefits of these processes towards improving quality assurance, findings show that adherence to Scrum processes remains a concern for many teams. On this point, participants believe that enforcement of Scrum practices such as code reviews, unit testing and close collaboration can reduce the current problems facing Scrum teams, including that of insufficient development time. In fact, insufficient development time seems to be a common concern that results in Scrum teams not being able to keep sprint delivery promises. For example, there were reported instances where work items (source code) were moved into the next sprint and it was later realized that all the required functionalities cannot be delivered to the customer at the end of a sprint .This moving of work items from one sprint to the next was cited as a common habit among teams. Recommendations on this point are presented under section 6.3 (recommendations) later in this chapter.

6.2.3 Measures of Product Quality

The adherence to some Scrum practices is based on various attributes, such as the team's choice, the project size and the development skills available, among others. It is therefore not easy to measure and compare quality assurance across the projects due to the fact that teams adhere to different practices. As mentioned several times before, some of the Scrum practices are not always necessary but there are other practices, such as close collaboration, which according to findings are essential to make a Scrum work. However, for teams to measure the level of customer satisfaction, they apply certain quality assurance measures. These include product demonstration to users, prior to signing off the product as complete, and User Acceptance Testing, when users accept that the system supports their needs. Findings suggest that a system demonstration serves two purposes, firstly it ensures that the customer is happy with the product being developed. Secondly, it ensures that developers give the users what they want in terms of meeting requirements. These are both measures of product quality. It is at this point that the development team gets another opportunity to better understand the needs of the customer. However, findings suggest, as one might expect, that this process works best in projects where requirements are fully understood. It emerged from the findings that there are times when users do not exactly know what they want, and as a result they cannot clearly explain what their needs are until they see a working product. Some participants confirmed that the only way that they measure quality is by looking at the business understanding, and by

making sure that all the project requirements are met. Once this point in the project has been reached they delegate the testing to the customer to verify the solution against their needs.

6.2.4 Success Rate of Scrum Projects

Reflecting on the definition of quality assurance as discussed in chapter two in section 2.1, in order to know whether the product meets acceptable quality standards, Scrum teams have to apply certain measures. Obviously, the intent of SQA measures is to assess the level of quality of a developed product and also to determine the success of a project. In terms of success rate, findings show that in spite of the challenges facing Scrum teams, such as insufficient development time, lack of resources (Scrum developers) and a lack of high skill levels, there has been a remarkable improvement. Scrum teams strive to deliver shippable and fully functional projects but conformance to standards is neglected. What stood out in the findings was that quality assurance is not a priority in Scrum teams. Teams choose when and which practices to apply in each project developed under Scrum methodology. However, in spite of the chaotic nature of the Scrum process, Scrum methodology simplifies the managements of projects for the Scrum master and project managers, especially in environments where requirements change often. This is an important point. When projects use Scrum rather than other methodologies, like waterfall methods, these projects can be delivered faster and the risk of failure (of the entire project) is minimal.

From the quality assurance and Scrum process perspective, the time factor was viewed as one of the aspects affecting product quality. On this point, findings reveal that time is the limiting factor. It is not the project that fails instead it is the implementation of the Scrum process and its application to the project. In this instance, the findings clearly pointed out that often times, Scrum success tends to be noticeable in teams where there is more than one team member who has the appropriate skills and who has worked in an Agile Scrum environment before.

6.2.5 Explanations to Quality Assurance Challenges in Scrum Implementation

Despite the notable significance and an increase in the adoption of Scrum methodologies compared to traditional methods such as waterfall, Scrum teams are still faced with various challenges, ranging from unclear requirements from users to the lack of developers' skill. There are various QA challenges facing Scrum teams but there was only one reported incident in this study where a project failed and had to be restarted. From a quality assurance perspective, the common challenge across Scrum teams was delivering projects on time, in other words, Scrum teams sometimes fail to deliver all the requirements on time. Therefore, this affects the project

budget, which is also a measure of quality assurance. It is evident beyond doubt from the findings that Scrum methods work best in environments where requirements change often and in small projects. However, its chaotic nature and flexibility remain a concern, especially for critical projects.

According to the findings, although users sometimes fail to clearly explain what they want, often times, it is not the project that fail but the process itself. Another aspect that came out of the literature and the findings was that Scrum does not prescribe quality assurance processes to be applied instead the processes used are based on the project in hand. Because Scrum quality assurance processes are not prescribed, they are difficult to measure.

Other common challenges to quality assurance include adaptability, lack of resources (including lack of developers' skill) and insufficient development time. On the adaptability aspect, according to the findings, some Scrum teams are facing challenges related to changing the way (traditional) team members work, when they are asked to move into agile methodologies. This also applies to other people in the organization, especially those who have been using traditional methods in developing software products. In this instance, findings show that although Scrum adoption is increasing, team members do not fully understand how to apply and implement Scrum methodology. Therefore, due to insufficient resources, Scrum teams end up not having enough development time. Delivery time was one of the most common challenges experienced by Scrum teams. Because of this problem, Scrum projects do not always finish on time remembering that success is measured by the delivery of a shippable fully functional product.

It is evident from the findings that most Scrum teams experience common challenges, for example problems pertaining to unclear requirements. Some of these problems are identified and fixed in the product demo session but it might be too late as this process is conducted when the requirements are already implemented.

6.3 Recommendations

Subsequent to the findings, recommendations can be summarised into four key points:

- Effecting the significance of quality assurance in Scrum
- Measures to effect the quality of software products
- Quality assurance and success rate of Scrum projects

- Related challenges to Scrum projects implementations

6.3.1 Effecting the significance of quality assurance in Scrum

The first point in the investigation was to ascertain the significance of quality assurance among Scrum teams. Findings suggest that the three most important quality assurance rules are code reviews, unit testing and user acceptance testing, and developers (especially their skills) are the most frequently recognized resources. They are central in the software development life cycle. The significance of these rules is to ensure that standards and procedures are used by resources in order to achieve a high level of product quality (in terms meeting the level of quality standards). It is therefore important for Scrum teams, especially team leaders, to enforce standards and consistent compliance with the rules, regardless of the limited development time and tight deadlines. However, participants argue that unit testing can waste time – especially when the code base is still small, but it is possible that source code can grow up to million lines. Therefore, sound practices should be enforced regardless of the project size. This is due to the fact that there are other benefits associated with unit testing such as auto-builds which also help during regression testing and integration testing, among others. Thus the standards associated with Code reviews should be enforced.

Further, though it is evident from the findings that even the most valued Scrum practices are applied optionally, strict compliance and adherence to frequent reviews are necessary to ensure that problems are found and addressed in the development phase while it is still easy and less expensive to detect defects. In other words, the project success rate must not only be measured by the delivery of a shippable functional product but also by the quality of the product and by the processes used to avoid high maintenance costs. From the theoretical perspective, in particular transformation of rules, technology changes every day therefore it is importance to review and amend standards to reflect the project needs in terms of quality and quality assurance planning should be included in the early phases of the project.

For example, there were reported instances where work items (source code) were moved into the next sprint and it was later realized that all the required functionalities cannot be delivered to the customer at the end of a sprint. This moving of work items from one sprint to the next was cited as a common habit among teams.

6.3.2 Measures to effect the quality of software products

On the measures of product quality – It seems that quality assurance measures vary according to the magnitude and size of the project, with most teams utilising the input of the clients to verify the adequacy of the product. What seems to be working well is to delegate the testing to the customer to verify the solution against their needs. An added value of this practice is that it ensures closer working relations between the Scrum production process and the client. As the saying goes "*don't fix it if it is not broken*", a recommendation is that as long as this practice yields satisfactory results, it should be encouraged and maintained.

6.3.3 Quality assurance and success rate of Scrum projects

On the aspect of a success rate of Scrum implementations, participants cited pressure to meet deadline – to the extent that adherence to standards tend to be neglected. Due to tight deadlines, most teams opt to take shortcuts. In other words, they do not follow standards rigorously. Further, limitations in terms of skills among Scrum team leaders also emerged as a concern. On the skill factor, insight among participants suggests that Scrum success tends to be noticeable in teams where there is more than one team member who has the appropriate skills and experience in an Agile Scrum environment therefore, a recommendation is that Scrum environments must ensure adequate skilled human-resources, mostly at leadership positions. On the point of pressure to meet deadlines, and therefore falling into the trap of short-cuts that neglects standards, it is strongly recommended in this thesis for team leaders to ensure that there are quality standards in place, and that code reviews are consistently enforced. In effect, it is the responsibility of team leaders to review these standards on a regular basis and ensure that developers comply with them. Finally, organizations must invest in resource development by making sure that developers attend Scrum implementation courses.

6.3.4 Related challenges to Scrum projects implementations

On challenges to Scrum processes - Meeting time frames on the delivery of a product was one common challenge cited by Scrum teams. The second most common challenge is that of inconsistency in terms of product adaptability of an application, and the willingness of developers to adapt to change. As a recommendation therefore, it is important for project product owners, project managers, the team leaders and business analysts to ensure that they meet with the user on a regular basis to verify requirements prior the implementation phase. In this instance, the use of prototypes can bridge the communication gap between the customer

and the development team. These prototypes must be presented and given to the customer for them to get the feel and look of their end product. This can then prompt feedback thus helping developers to build the right product that satisfies the user. Having active stakeholder involvement can minimize development costs and time.

6.4 Suggestion for Future Research

This study only focused on five Scrum teams and two academic institutions. Due to time constraints, the researcher could not manage to interview all the possible teams. Participants who took part in this study were from three South African provinces: Eastern Cape, Gauteng and Western Cape, leaving out six provinces. Another study could explore how quality assurance standards are enforced in SDLC across nine provinces and other academic institutions. This could provide a comparison of Scrum implementation and success. This study focused on how Scrum teams can understand and maximize the quality of projects developed under Scrum methodology in distributed teams. On this point another study could conduct a follow up research to explore projects from the users' perspective when the product is in operation. Such a study could also look at maintainability, adaptability and whether or not the system meets users' needs. Another study could look at how organizations can enforce compliance to software quality assurance standards and procedures to improve product quality.

6.5 Conclusion

This chapter concludes this study, reflecting on the value of the thesis and illustrating how the study's objectives have been met. It started with a summary of the findings providing an overview on the purpose of the study, the questions asked and the research methods used to answer the research question. In closing, the chapter presented the recommendations to the findings and the suggestions for future studies.

7. References

Abrahamsson, P., Conboy, K. & Wang, X., 2009. “Lots done, more to do”: the current state of agile systems development research. *European Journal of Information Systems*, 18(4), pp.281–284. Available at: http://www.palgrave-journals.com/doifinder/10.1057/ejis.2009.27 [Accessed April 11, 2014].

Advoss, 2014. Software Quality Attributes. Available at: http://www.advoss.com/software-quality-attributes-customizability.html [Accessed November 15, 2014].

Aggarwal, K.K. & Sigh, Y., 2001. *Software Engineering programs documentation, operating procedures* 2nd ed., New Delhi: New Age International Publishers.

Agile Development, 2011. Reflections on Agile 2011. Available at: http://agile-development.biz/tag/scum-complexity/.

Ahamed, R., 2011. The Impact of Formal Approaches to Software Quality. *Computer Science and telecommunication*, 1(30), pp.14–20.

Ahmed, A. et al., 2010. Agile software development: Impact on productivity and quality. *2010 IEEE International Conference on Management of Innovation & Technology*, pp.287–291. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5492703.

Al-azzah, F.M. & Yhya, A.A., 2011. Quality Criteria ’ s of Computing Disciplines. *European Journal of Scientific Research*, 48(3), pp.352–360.

Alsultanny, Y. a. & Wohaishi, A.M., 2009. Requirements of Software Quality Assurance Model. *2009 Second International Conference on Environmental and Computer Science*, pp.19–23. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5383564 [Accessed September 14, 2013].

Amagoh, F., 2008. Perspectives on Organizational Change: Systems and Complexity Theories. *The Public Sector Innovation Journal*, 13(3), pp.1–14.

Ambler, S., 2005. Quality in an agile world. *Software Quality Professional*, 7(3). Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.7560&rep=rep1&type=pdf [Accessed September 14, 2013].

Andersson, C., 2003. *Exploring the Software Verification and Validation Process*. Lund University.

Bae, B. & Ashcroft, P., 2004. Implementation of ERP Systems : Accounting and Auditing Implications. *Information Systems Control Journal*, 5, pp.43–48.

Balaji, S., 2012. Waterfall vs V-Model vs Agile: A Comparative study on SDLC. *International Journal of Information technology and Business Management*, 2(1), pp.26–30.

Banker, R., Datar, S. & Zweig, D., 1989. Software Complexity and Maintainability. , 11(5.6), p.3.

Barbie, E. & Mouton, J., 2001. *The Practice of Social Research*, Cape Town: Oxford University Press.

Barratt-Pugh, L., 2007. Exploring and evaluating Structuration Theory as a framework for investigating formal and informal learning within organisations .

Bender RBT Inc, 2003. Systems Development Life Cycle : Objectives and Requirements.

Berander, P. et al., 2005. *Software quality attributes and trade-offs*. Blekinge Institute of Technology.

Berends, H., Boersma, K. & Weggeman, M., 2003. The Structuration of Organizational Learning. *Human relations*, 56(9), pp.1035–1056.

Bevan, N., 1995. Measuring usability as quality of use. *Software Quality journal*, 4, pp.115–150.

Bevan, N., 1997. Quality and usability : A new framework. , (figure 1), pp.1–8.

Bevan, N., 1999. Quality in use: Meeting user needs for quality. *Journal of Systems and Software*, pp.1–14. Available at: http://www.sciencedirect.com/science/article/pii/S0164121299000709 [Accessed August 12, 2013].

Bhasin, S., 2012. Quality Assurance in Agile: A Study towards Achieving Excellence. *2012 Agile India*, pp.64–67. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6170012 [Accessed March 18, 2013].

Bouthillier, F., 2000. The Meaning of Service : Ambiguities and Dilemmas for Public Library Service Providers. *Library & Information Science Research*, 22(3), pp.243–272.

Bracken, S., 2014. Discussing the Importance of Ontology and Epistemology Awareness in Practitioner Research. *Worcester Journal of Learning and teaching*, 4, pp.1–9.

BSSC, 1995. Guide to software quality assurance. , (1).

Burke, M.E., 2007. Making choices: research paradigms and information management: Practical applications of philosophy in IM research. , 56(6), pp.476–484.

Business Dictionary, 2014a. Complexity. Available at: http://www.businessdictionary.com/definition/complexity.html.

Business Dictionary, 2014b. Research Design. Available at: http://www.businessdictionary.com/definition/research-design.html [Accessed March 5, 2014].

Business Disctionary, 2013. Quality. Available at: http://www.businessdictionary.com/definition/quality.html.

Cadle, J. & Yeates, D., 2008. *Project Management for Information Systems (5th Edition)* 5th ed., Harlow: Pearson Education Limited.

Cameron, R., 2011. Quality frameworks and procedural checklists for mixed methods research.

Chan, S., 2001. Complex Adaptive Systems. *ESD.83 Research Seminar in Engineering Systems*, pp.1–9.

Chegg, 2014. Definition of Functionalism. Available at: http://www.chegg.com/homework-help/definitions/functionalism-49.

Chung, L. & do Prado, L., 2009. On Non-Functional Requirements in Software. , pp.363–379.

Clear, T. & MacDonell, S.G., 2011. Understanding technology use in global virtual teams: Research methodologies and methods. *Information and Software Technology*, 53(9), pp.994–1011. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0950584911000395 [Accessed March 24, 2014].

Cline, A., 2014. What is Epistemology? Philosophy of Truth, Knowledge, Belief. Available at: http://atheism.about.com/od/philosophybranches/p/Epistemology.htm.

Cockburn, A., 2001. Agile Software Development. , pp.2000–2001.

Collar, E. & Valerdi, R., 2006. Role of Software Readability on Software Development Cost. *21st Forum on COCOMO and Software Cost Modeling*.

Conklin, W.A., 2011. Software Assurance: The Need for Definitions. *2011 44th Hawaii International Conference on System Sciences*, pp.1–7. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5718650.

Coreil, J., 2010. Behavioral and Social Science Theory. In *Social and behavioral foundations of public health*. pp. 101–114.

Creswell, J.W., 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* 2nd ed., Califonia: SAGE Publications Inc.

Crossman, A., 2014. Content Analysis. Available at: http://sociology.about.com/od/C_Index/g/Content-Analysis.htm.

CS, 1995. SCRUM Planning Phase. , pp.1–7. Available at: http://www.cs.uu.nl/wiki/bin/view/Spm/ScrumPlanningPhase#descr.

Current Nursing, 2013. Development of Nursing Theories. Available at: http://currentnursing.com/nursing_theory/development_of_nursing_theories.html [Accessed October 10, 2014].

Czarnacka-Chrobot, B., 2010. The Economic Importance of Business Software Systems Size Measurement. *2010 Fifth International Multi-conference on Computing in the Global Information Technology*, pp.293–299. Available at:

http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5628920 [Accessed May 4, 2013].

DeSanctis, G. & Poole, M.S., 1994. Capturing the Complexity in Advanced Technology use: Adaptive Structuration Theory. *Organization Science*, 5(2), pp.121–147.

Dictionary, 2014. Typology. Available at: http://dictionary.reference.com/browse/typology [Accessed July 10, 2014].

Dillard, J. & Pullman, M., 2009. A Structuration Frame for Social Enterprise and an agricultural example. *Economia Aziendale Online*, 1(1), pp.89–108.

Dingsoyr, T. et al., 2012. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), pp.1213–1221. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0164121212000532 [Accessed April 28, 2014].

Dingsoyr, T., Dyba, T. & Abrahamsson, P., 2008. A Preliminary Roadmap for Empirical Research on Agile Software Development. *Agile 2008 Conference*, pp.83–94. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4599455 [Accessed March 19, 2014].

Dyba, T. & Dingsoyr, T., 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50, pp.833–859. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0950584908000256 [Accessed March 19, 2014].

Easterbrook, S., 2010. The difference between Verification and Validation. Available at: http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/.

Edelson, D.C., 2002. Design Research: What We Learn When We Engage in Design. *Journal of the Learning Sciences*, 11(1), pp.105–121. Available at: http://www.tandfonline.com/doi/abs/10.1207/S15327809JLS1101_4.

Eden, a. H. & Mens, T., 2006. Measuring software flexibility. *IEE Proceedings - Software*, 153(3), pp.113–126. Available at: http://digital-library.theiet.org/content/journals/10.1049/ip-sen_20050045.

Eidelson, R.J., 1997. Complex adaptive systems in the behavioral and social sciences. *Review of General Psychology*, 1(1), pp.42–71. Available at: http://doi.apa.org/getdoi.cfm?doi=10.1037/1089-2680.1.1.42.

Elam, D., 2011. Fit for Purpose : Planning and Communicating project quality requirements.

Engel, A. & Browning, T.R., 2006. Designing Systems for Adaptability by Means of Architecture Options. *System Engineering: Shining Light on the Tough Issues*.

English, L., 1996. Information quality: Meeting customer needs. *Information Impact Newsletter*, 3(1), pp.1–9. Available at: http://connect.msbcollege.edu/bbcswebdav/pid-1637171-dt-content-rid-31106_1/library/Graduate/TM520/Additional Resources/Meeting Customer Needs.pdf [Accessed August 12, 2013].

eSource Research, 2014. Theory and Why It is important. Available at: http://www.esourceresearch.org/eSourceBook/SocialandBehavioralTheories/3TheoryandWhyItisImportant/tabid/727/Default.aspx.

Ewusi-Mensah, K., 2003. *Software Development Failures: Anatomy of Abandoned Projects*, Cambridge: Mit Press.

Faegri, T.E., 2011. *Collaborative learning in software development: An investigation of characteristics, prerequisites and improvement*. Norwegian University of Science and technology.

Felix, T. & Sedera, G., 2007. Conceptualizing Interaction With ERP Systems Usig Adaptive Structuration Theory.

Fereday, J. & Muir-cochrane, E., 2006. Demonstrating Rigor Using Thematic Analysis : A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods*, 5(1), pp.1–11.

Fetaji, B. & Fetaji, M., 2009. Software Solution Using Task Based Learning Approach. , pp.395–399.

Fitzpatrick, R., 1996. *Software Quality: Definitions and strategic issues*,

Flowers, P., 2006. Research Philosophies – Importance and Relevance. , 1(1), pp.1–5.

Fong, A., Valerdi, R. & Srinivasan, J., 2007. Using a Boundary Object Framework to Analyze Interorganizational Collaboration.

Forrest, S. & Jones, T., 1995. Modeling Complex Adaptive Systems with Echo. *Complex Systems: Mechanisms of Adaptation*, pp.1–21.

Fowler, M. & Highsmith, J., 2001. The Agile Manifesto. *Software Development*, 9(8), pp.28–35.

Freeman, E. et al., 2004. *Head First Design Patterns*, O'Reilly.

Galin, D., 2004. *Software quality assurance: from theory to implementation*, Harlow: Pearson Education Limited. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Software+Quality+Assurance:+From+theory+to+implementation#0 [Accessed June 19, 2013].

GAO, 1992. *Quantitative Data Analysis: An Introduction*,

Gay, B. & Weaver, S., 2011. Theory Building and Paradigms : A Primer on the Nuances of Theory Construction. *American Internation Journal of Contomporary Research*, 1(2), pp.24–32.

Geambasu, V.C. et al., 2011. Influencing Factors For The Choice of A Software Development Methodology. *Accounting & Management Information Systems/Contabilitate si Informatica de Gestiune*, 10(4), pp.479–494.

Giddens, A., 1989. *Social Theory of Modern Societies: Anthony Giddens and his Critics* D. Held, ed., Cambridge: Press Syndicate of the University of Cambridge.

Giddens, A., 1984. *The Constitution of Society: Outline of the Theory of Saturation*, University of Califonia Press.

Goldreich, O., 2005. A brief overview of Complexity Theory. Available at: http://www.wisdom.weizmann.ac.il/~oded/cc.html.

Gotelli, N.J. & Llison, A.M., 2004. *A Primer of Ecological Statistics* illustrate., Sinauer Associates.

Gregor, S., 2002. A Theory of Theories in Information Systems. , pp.1–18.

Hamman, M., 2009. Organizational Agility Capality Theory, and Social Practices.

Harwell, M., 2011. Research Design in Qualitative/Quantitative/Mixed Methods. *CF Clifton and*, pp.147–182. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Research+Design+In+Qualitative/Quantitative/Mixed+Methods#4 [Accessed September 18, 2013].

Hirschheim, R., 1985. Information systems epistemology: An historical perspective. *Research methods in information systems*. Available at: http://ifipwg82.org/sites/ifipwg82.org/files/Hirschheim_0.pdf [Accessed March 18, 2014].

Holtta, V., 2013. *Beyond Boundary Objects - Improving Engineering Communication with Conscription Devices*. Aalto University.

Hsieh, H. & Shannon, S.E., 2005. Three Approaches to Qualitative Content Analysis. *Qualitative Health Research*, 15(9), pp.1277–1288.

Huo, M. et al., 2004. Software quality and agile methods. *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pp.520–525. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1342889.

Hussey, J. & Hussey, R., 1997. *Business Research: A Practical Guide for Undergraduate and Postgraduate Students*, Mcmillan Publisher Limited.

Ichu, E. & Nemani, R., 2011. The Role of Quality Assurance in Software Development Projects: Project Failures and Business Performance. *International Journal of Computer Technology and Applications*, 2(4), pp.716–725.

Imreh, R. & Raisinghani, M.S., 2011. Impact of Agile Software Development on Quality within Information Technology Organizations. *2009-2011 CIS Journal*, 2(10), pp.460–475.

Innolance, 2013. Our Development Process. Available at: http://www.innolance.com/process.

Jagun, A. & Heeks, R., 2007. Mobile phones and development: The future in new hands? *Id21 Insights*, 69.

Jeffery, A., 2003. Using structuration theory to make sense of requirements practice in a small software business. , pp.57–65.

Jensen, P.A., 2004. Models. , pp.1–2. Available at: http://www.me.utexas.edu/~jensen/ORMM/models/.

Jirava, P., 2004. System Development Life Cycle. , pp.58–62.

Johnson, P., Ekstedt, M. & Jacobson, I., 2012. Where's the Theory for Software Engineering? *IEEE Software*, 29(5), p.96.

Jones, C. & Bonsignour, O., 2012. *The economics of software quality* B. Goodwin, ed., Boston: Addison-Wesley Professional. Available at: http://books.google.com/books?hl=en&lr=&id=_t5l5Cn0NBEC&oi=fnd&pg=PR9&dq=The+Economics+of+Software+Quality&ots=8RkUVnyqqa&sig=ZRGg2NnlMvMLcRZhPtlRH1YDzzA [Accessed August 12, 2013].

Jones, M. & Karsten, H., 2003a. Management Studies Information Systems Research.

Jones, M. & Karsten, H., 2003b. Review: Structuration Theory and Information Systems Research. *Judge Institute of Management Working Paper*, 11.

Kawulich, B., 2004. Data Analysis Techniques in Qualitative Research. *Journal of Research in Education*. Available at: http://kimhuett.wiki.westga.edu/file/view/kawulich_2004.pdf [Accessed February 20, 2014].

Kawulich, B.B., 2005. Participant Observation as a Data Collection Method. , 6(2).

Kayes, I., 2011. Agile Testing: Introducing PRAT as a metric of testing quality in Scrum. *ACM SIGSOFT Software Engineering Notes*, 36(2), pp.1–5.

Kearney, J.K. et al., 1986. Software complexity measurement. *Communications of the ACM*, 29(11), pp.1044–1050.

Keith, M., 2006. Complexity Theory and Education. *APERA Conference*, pp.1–12.

Kemerer, C., 2009. The Impact of Design and Code Reviews on Software Quality : An Empirical Study Based on PSP Data. *IEEE Transactions on Software Engineering*, 35, pp.1–17.

Kent, E., 2013. New Research Provides Insight Into Women's Use of Mobile Phones. Available at: http://socialmediatoday.com/elizabethkent/1898226/new-research-women-mobile-phones-usage.

Khalaf, J. & Al-Jedaiah, M.N., 2008. Software quality and assurance in waterfall model and XP - a comparative study. *WSEAS Transactions on Computers*, 7(12), pp.1968–1976.

Khalane, T., 2013. *Software Quality Assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations*. University of Cape Town.

Khraiwesh, M. & Jordan, Z., 2011. Validation Measures in CMMI. *World of Computer Science and Information technology*, 1(2), pp.26–33.

King, A., 2012. *The Structure of Social Theory*, Canada: Routledge.

Kitchenham, B. & Pfleeger, S.L., 2002. Principles of Survey Research Part 5 : Populations and Samples. , 27(5), pp.17–20.

Kitchenham, B. & Pfleeger, S.L., 1996. S o f t w a r e quality: The Elusive Target.

Knippers, D., 2011. Agile Software Development and Maintainability.

Kokar, M.M., Baclawski, K. & Eracar, Y. a., 1999. Control Theory-Based Foundations of Self-Controlling Software. *IEEE Intelligent Systems*, 14(3), pp.37–45. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=769883.

Kondracki, N., Wellman, N. & Amundson, D., 2003. Content Analyis: Review of Methods and Their Applications in Nutrition Education. , 34, pp.224–230.

Koskela, L. & Howell, G., 2002. The Theory of Project Management: Aexplanation to Novel Methods. *Proceedings IGLC*, pp.1–11.

Krauss, S.E. & Putra, U., 2005. Research Paradigms and Meaning Making : A Primer. , 10(4), pp.758–770.

Kulas, H., 2012. *Product metrics in agile software development*. University of Tampere.

Leau, Y. et al., 2012. Software Development Life Cycle AGILE vs Traditional Approaches. *2012 International Conference on information and Network Technology (ICINT 2012)*, 37, pp.162–167.

Liong, M., 2004. *Software Development Lifecyle*, Pearson Education Limited. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Practical+Software+Engineering#0 [Accessed June 17, 2013].

Loureiro-koechlin, C., 2008. A Theoretical Framework for a Structuration Model of Social Issues in Software Development in Information Systems. *Systems Research and Behavioral Science*, 109(1), pp.99–109.

Loureiro-Koechlin, C., 2008. A theoretical framework for a structuration model of social issues in software development in information systems. *Systems Research and Behavioral Science*, 25(1).

Lyytinen, K. j & Ngwenyama, O.K., 1992. What does computer support for cooperative work mean? A structurational analysis of computer supported cooperative work. , 2(I), pp.19–37.

Ma, L., 2010. Theory and Education : A Case of Structuration Theory.

Mack, L., 2010. The Philosophical Underpinnings of Educational Research. , 19, pp.5–11.

Mahanti, A., 2007. Challenges in Enterprise Adoption of Agile Methods - A Survey. *Journal of Computing and Information Technology*, 14(3), pp.197–206. Available at: http://cit.zesoi.fer.hr/browsePaper.php?paper=752.

Makabee, H., 2013. Effectice Software Design. Available at: http://effectivesoftwaredesign.com/2013/05/05/adaptable-designs-for-agile-software-evolution/ [Accessed December 10, 2014].

Mansor, Z., Yahya, S. & Arshad, N.H., 2011. Towards the Development of Success Determinants Charter for Agile Development Methodology. *International Journal of Information technology and Engineering*, 2(1), pp.1–7.

Manuel, R., 1999. *The Organizational Implementation of Information Systems : towards a new theory The London School of Economics*. School of Economics.

Maruping, L.M., Venkatesh, V. & Agarwal, R., 2009. A Control Theory Perspective on Agile Methodology Use and Changing User Requirements. *Information Systems Research*, 20(3), pp.377–399. Available at: http://pubsonline.informs.org/doi/abs/10.1287/isre.1090.0238 [Accessed March 24, 2014].

Massey, V. & Satao, K.J., 2012. Evolving a New Software Development Life Cycle Model (SDLC) incorporated with Release Management. *Internation Journal of Engineering and Advanced Technology*, 1(4), pp.25–31.

Mccormick, M., 2012. Waterfall vs. Agile Methodology.

Mchunu, N.N., 2013. *Adequacy of healthcare information systems to support data quality in the public healthcare sector , in the Western Cape , South Africa by Nokubalela Ntombiyethu Mchunu Thesis submitted in fulfilment of the requirements for the degree Master of Technology*. Cape Peninsula University of technology.

Meli, R., 1999. Risks , requirements and estimation of a software project. , pp.1–14.

Mertens, D.M., 2010. P hilosophy in mixed methods teaching : The transformative paradigm as illustration. *International Journal of Multiple Research Approach*, 4(1), pp.9–18.

Meso, P. & Jain, R., 2006. Agile Software Development: Adaptive Systems Principles and Best Practices. *Information Systems Management*, 23(3), pp.19–30.

Miller, J.H. & Page, S.E., 2007. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life* S. A. Levin & S. H. Strogaz, eds., Oxfordshire: Princeton University Press. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24815723.

Mlitwa, N., 2011. *Integration of e-learning Systems into Academic Programmes in Modern Universities: A South African Perspective*, Cape Town: TVK e-NNOVATIONS.

Mnkandla, E. & Dwolatzky, B., 2006. Defining Agile Software Quality Assurance. *Internation Conference on Software Engineering Advances*.

Mohammad, A.H., 2013. Agile Software Methodologies : Strength and Weakness. *International Journal of Engineering Science and Technology (IJEST)*, 5(03), pp.455–459.

Moniruzzaman, A. & Hossain, D., 2013. Comparative Study on Agile software development methodologies. *arXiv preprint arXiv:1307.3356*. Available at: http://arxiv.org/abs/1307.3356 [Accessed November 8, 2013].

Morril et al., 2000. Qualitative data analysis. In p. 521. Available at: http://www.rds-yh.nihr.ac.uk/wp-content/uploads/2013/05/9_Qualitative_Data_Analysis_Revision_2009.pdf [Accessed March 18, 2014].

Mosaicinc, 2001. What Is The Difference Between Quality Assurance, Quality Control, And Testing? Available at: http://www.mosaicinc.com/mosaicinc/rmThisMonth.asp.

Mouton, J., 1996. *Understanding Social Research* 1st ed., Pretoria: Van Schaik Publishers.

Nafees, T., 2011. Impact of user satisfaction on Software quality in use. *International Journal of Electrical & Computer Sciences …*, 11(03), pp.48–56. Available at: http://ijens.org/Vol 11 I 03/118003-2929 IJECS-IJENS.pdf [Accessed August 12, 2013].

Nagy, S., Hesse-Biber & Patricia, L., 2011. *The Practice of Qualitative Research* 2nd ed., Boston: Sage Publication Inc.

Naidoo, T.-R., 2009. *Towards a conceptual framework for understanding the implementation of Internet-based self-service technology*. University of Pretoria.

Naik, K. & Tripathy, P., 2008. *Software testing and quality assurance: theory and practice*, New jersey: John Wiley & Sons, Inc. Available at: http://books.google.com/books?hl=en&lr=&id=neWaoJKSkvgC&oi=fnd&pg=PT13&dq=Software+testing+and+quality+assurance:+Theory+and+practice&ots=_bsgOIxXjr&sig=_2OdCnXQ5wGhqaGZDsAKmPtEPG4 [Accessed June 11, 2013].

NASA, 2004. Software Assurance Standard.

Nawaz, A. & Malik, K., 2008. *Software Testing Process in agile development*. Blekinge Institute of Technology.

NCSE, 2012. Definitions of Fact , Theory , and Law in Scientific Work. , pp.2–3. Available at: http://ncse.com/evolution/education/definitions-fact-theory-law-scientific-workhttp://ncse.com/evolution/education/definitions-fact-theory-law-scientific-work.

Nerur, S., Mahapatra, R. & Mangalaraj, G., 2005. Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), pp.72–78. Available at: http://portal.acm.org/citation.cfm?doid=1060710.1060712.

Neuman, L., 2006. *Social Research Methods: Qualitative and Quantitative Approaches* 6th ed., Boston: Pearson.

Onwuegbuzie, A.J., Leech, N.L. & Collins, K.M.T., 2012. Qualitative Analysis Techniques for the Review of the Literature. *The Qualitative Report 2012*, 17, pp.1–28.

Orlikowski, W.J., 1991. The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3(3141).

Orlikowski, W.J., 2000. Using Technology and Constituting Structures : Practice Lens in for Studying Technology Organizations. *Organization science*, 11(4), pp.404–428.

OSQA, 2009. SDLC Models. Available at: info@onestopqa.com.

Overhage, S. & Schlauderer, S., 2012. Investigating the Long-Term Acceptance of Agile Methodologies: An Empirical Study of Developer Perceptions in Scrum Projects. *45th Hawaii International Conference on System Sciences*, pp.5452–5461. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6149555 [Accessed April 11, 2014].

Owens, D.M. & Khazanchi, D., 2009. Software Quality Assurance. , pp.245–263.

Oxford Dictionary, 2014a. Complexity. Available at: http://www.oxforddictionaries.com/definition/english/complexity.

Oxford Dictionary, 2014b. Knowledge.

Parasoft, 2010. Use Coding Standards to Increase Efficiency and Productivity.

Parker, J., 2010. Structuration Theories. *Historical Developments and Theoretical Approaches in Sociology*, 2.

Petersen, K., Wohlin, C. & Baca, D., 2009. The Waterfall Model in Large-Scale. , pp.386–400.

Pettigrew, K.E. & McKechnie, L., 2001. The Use of Theory in Information Science Research. *Journal of the American Society for Information Science and Technology*, 52(1), pp.62–73.

Post, D.E. & Kendall, R.P., 2004. Software Project Management and Quality Engineering Practices for Complex, Coupled Multi-Physics, Massively Parallel Computational Simulations: Lessons Learned from ASCI. , pp.1–27.

Pressman, R.S., 2010. *Software Engineering: A Practitioner's Approach* 7th ed., McGraw-Hill.

Pries-heje, L. & Pries-heje, J., 2011. Why Scrum works: A case study from an agile distributed project in Denmark and India. *Agile Conference (AGILE)*, pp.20–28.

Quizlet, 2014. Structuration Theory. Available at: http://quizlet.com/8147989/structuration-theory-flash-cards/.

Rajasekar, S., Philamnothan, P. & Chinnathambi, V., 2006. Research methodology. , (1), pp.1–23. Available at: http://link.springer.com/content/pdf/10.1007/0-387-23273-7_3.pdf [Accessed September 18, 2013].

Ralph, P., 2013. Possible Core Theories for Software Engineering. *2nd Workshop on a General Theory of Software Engineering*, pp.35–38.

Reeves, S. et al., 2008. Why use theories in qualitative research? , 337, pp.1–4.

Rising, L. & Janoff, N., 2000. The Scrum software development process for small teams. *Software, IEEE*. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=854065 [Accessed September 14, 2013].

Rohil, H., 2012. Analysis of Agile and Traditional Approach for Software Development. , 1(4), pp.1–10.

Ropa, S., 2014. Agile Software Programming Best Practices. Available at: http://www.versionone.com/Agile101/Agile-Software-Programming-Best-Practices/.

Rose, J., 1998. Structuration Theory and Information System Development - Framework for Practice. , (Rose).

Rose, J. & Hackney, R., 2002. Towards a Structurational Theory of Information Systems : a Substantive Case Analysis. *Proceedings of the 36th Annual Hawaii International Conference*.

Rose, J. & Hackney, R., 2003. Towards a Structurational Theory of Information Systems : a Substantive Case Analysis. , 00, pp.1–9.

Rose, J. & Scheepers, R., 2001. Structuration Theory and Information System Development - Framework For Practice. *The 9th European Conference on Information Systems (ECIS)*, pp.217–231.

Rosenbaum, H., 2010. A Structuration Approach to Online Communities of Practice : The Case of Q&A Communities. *Journal of the American Society for Information Science and Technology*, 61(9), pp.1933–1944.

Safecode, 2008. An Overview of Current Industry Best Practices Software Assurance : Executive Summary.

Sallis, E., 1996. *Total quality management in education*, Available at: http://books.google.com/books?hl=en&lr=&id=QAOORZ9NdHQC&oi=fnd&pg=PP2&dq=Total+Quality+Management+in+education&ots=YtfvJMXeJZ&sig=IFFS0n7ZjhmZQHUH8RfG6o6Ozyk [Accessed August 6, 2013].

Salo, O. & Abrahamsson, P., 2006. An Iterative Improvement Process for Agile Software Development. *Software Process: Improvement and Practice*, 12(1), pp.81–100.

Santamaria, M.G., 2007. Agile & Scrum : What are these methodologies and how will they impact QA / testing roles ?

Schwaber, K., 2004. *Agile Project Management with Scrum*, Redmond, Washington: Microsoft Press.

Schwaber, K., 1994. SCRUM Development Process. , pp.10–19.

Select Business Solutions, 2013. What is the Waterfall Model? Available at: http://www.selectbs.com/analysis-and-design/what-is-the-waterfall-model.

Serena, 2007. *An Introduction to Agile Software Development*, Available at: www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf.

Serumgard, S., 1997. *Verification of Process Conformance in Empirical Studies of Software Development*. Norwegian Univeristy of Science and Technology.

Sfetsos, P. & Stamelos, I., 2010. Empirical Studies on Quality in Agile Practices: A Systematic Literature Review. *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pp.44–53. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5654783 [Accessed April 11, 2014].

Shen, L. & Ren, S., 1990. Analysis and measurement of software flexibility based on flexible points. , pp.331–341.

Sirshar, M. & Arif, F., 2012. Evaluation of Quality Assurance Factors in Agile Methodologies. *International Journal of Advanced Computer Science*, 2(2), pp.73–78.

Smith, L.W., 2003. Software Life Cycle. In *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems*. pp. 1–14.

Sneed, H.M. & Merey, a., 1985. Automated Software Quality Assurance. *IEEE Transactions on Software Engineering*, SE-11(9), pp.909–916. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1702108.

SoftWaysSolutions, 2012. There Might Be Scrum-thing To This. Available at: http://www.softwaysolutions.com/blog/might-scrum-thing/ [Accessed October 10, 2014].

Sohaib, O. & Khan, K., 2010. The Role of Software Quality in Agile Software Development Methodologies. *Journal of Engineering and sciences*, pp.20–25.

Sommerville, I., 2011. *Software Engineering* 9th ed. M. Horton et al., eds., United States of America: Addison-Wesley.

Sousa, D., 2013. The Advantages and Disadvantages of Agile Software Development. Available at: http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-agile-software-development.html [Accessed January 9, 2015].

Spradley, J., 1980. Participant observation. Available at: http://www.citeulike.org/group/8921/article/4046568 [Accessed February 13, 2014].

Stillman, L. & Stoecker, R., 2005. Structuration , ICTs , and Community Work. *The Journal of Cummunity Imformatics*, 1(3), pp.83–102.

Stray, V.G., 2010. A Comparative Case Study of Teamwork in Norwegian Agile Software Development Projects.

Subramaniam, H. & Zulzalil, H., 2012. Software Quality Assessment Using Flexibility: a Systematic Literature Review. *International Review on Computers and Software*, 7(5).

Subramanian, N. & Chung, L., 2001. Software Architecture Adaptability : An NFR Approach. , pp.1–10.

Sutherland, J., 2004. Agile Development: Lessons Learned From First Scrum Agile Development.

Szalvay, V., 2004a. An Introduction to Agile Software Development. *Danube technologies*, pp.1–9.

Szalvay, V., 2004b. An Introduction to Agile Software Development. *Danube Technologies*, pp.1–9.

Taylor-powell, E., 2003. Analyzing Quantitative Data.

Tayntor, C., 2002. *Six Sigma Software Development*, United States of America: CRC Press.

TCS, 2013. *Scrum Manager*,

TechTarget, 2013. Agile Software Development (ASD). Available at: http://searchsoftwarequality.techtarget.com/definition/agile-software-development.

Teddlie, C., 2006. A General Typology of Research Designs Featuring Mixed Methods 1. , 13(1), pp.12–28.

The Free Dictionary, 2013. Software Quality Assurance.

The Software Development Life Cycle (SDLC), 2005. The Software Development Life Cycle (SDLC). , pp.1–24.

Theng, Y. et al., 2008. Claims Analysis Meets Structuration Theory : Analysing Qualitative Students“ Interactions with NTU’s edveNTUre Review of Selected, Established Theories for Analysing User Interactions NTU”s edveNTUre. *World Conference on Education Multimedia, Hypermedia and telecommunication*.

Thorne, S., Kirkham, S.R. & Flynn-magee, K.O., 2004. The Analytic Challenge in Interpretive Description. *International Journal of Qualitative Methods*, 3(1), pp.1–21.

Timperi, O.P., 2004. An Overview of Quality Assurance Practices in Agile Methodologies. *T-76.650 Seminar in Software Engineering*.

Tomar, A. & Thakare, V.M., 2012. Identification and Listing of Factors Affecting Software Quality Assurance. , pp.43–47.

Truren, B., 2010. *Improving software flexibility in a smart business network*. Delft University of Technology.

Tuteja, M. & Dubey, G., 2012. A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models. *Internation Journal of Soft Computing and Engineering (IJSCE)*, 2(3), pp.251–257.

Uddin, M.N. & Hamiduzzaman, M., 2009. The Philosophy of Science in Social. *Journal of International Social Research*, 2(6), pp.131–132.

Udo-akang, D., 2012. Theoretical Constructs , Concepts , and Applications. *Merical International Journal of Contemporary Research*, 2(9), pp.89–97.

Ullah, M.I. & Zaidi, W.A., 2009. *Quality Assurance Activities in Agile - Philosophy to Practice*. Blekinge Institute of Technology.

Underwood, K. et al., 2013. *Guide to Business Continuity Management*, United Kingdom.

Veenstra, A.F. van, Melin, U. & Axelsson, K., 2014. Theorectical and Practical Implications from the use of Structuration Theory in Public Sector Information Systems Research. *European Conference of Information Systems (ECIS)*, pp.1–12.

Wallin, C., 2002. *Verification and Validation of Software Components and Component Based Software Systems*,

Walsham, G., 1995. Interpretive case studies in IS research : nature and method. , 4.

Wangler, B. & Backlund, A., 2005. Information Systems Engineering : What Is It? *CAiSE Workshops*, pp.427–437.

Webopedia, 2014. Software Quality Assurance. Available at: http://www.webopedia.com/TERM/S/Software_Quality_Assurance.html [Accessed May 10, 2014].

Williams, L. et al., 2011. Scrum + Engineering Practices : Experiences of Three Microsoft Teams. *Empirical Software Engineering and Measurement (ESEM)*, pp.463–471.

Wilson, D.N. & Hall, T., 1998. Perceptions of software quality : a pilot study. *Software Quality Journal*, 7, pp.67–75.

Winter, J. et al., 2008. Meeting Organisational Needs and Quality Assurance through Balancing Agile & Formal Usability Testing Results. *Software Engineering Techniques. Springer Berlin Heidelberg*, pp.275–289.

Wolfel, R.L., 2005. Migration in the New World Order: Structuration Theory and its contribution to Explanations of Migration. *Geography Online*, 5(2).

Worrall, A., Boundary Object Theory: Origins, Development, and Applications.

Yahaya, J.H., Deraman, A. & Hamdan, A.R., 2008. Software Quality from Behavioural and Human Perspectives. *Internation Journal of Computer Science and Network Security*, 8(8), pp.53–63.

Yates, J. & Giddens, A., 1997. Using Giddens ' Structuration Theory to Inform Business History theorist. *Business and Economic History*, 26(1), pp.159–183.

Younga, R., 2002. Recommended requirements gathering practices. *CrossTalk*, pp.9–12. Available at: http://staff.unak.is/andy/Year2 Object Oriented Methods/HomeworkTests/RecReqGatPra.pdf [Accessed June 17, 2013].

Zhang, Y. & Wildemuth, B.M., 2005. Qualitative Analysis of Content. *Applications of social research methods to questions in information and library science*, pp.308–319.

Zimmerman, K.A., 2012. What is Social Theory. Available at: http://www.livescience.com/21491-what-is-a-scientific-theory-definition-of-theory.html.

8. Appendices

The aim of this section is to present the annexes, it consists of six annexes with data transcripts in annexure 1, followed by research ethics letters in annexure 2. In annexure 3 is the research interview request letter, annexure 4 interview questionnaire sample. The Agile Scrum project management process showing a graphical representation of a Scrum process is shown in Figure 4.

Annexure 1: Data Transcripts

Date: Friday, 26 May 2014 09:15 – 09:45

Interviewer: Andile Koka

Interviewee: [redacted] – Senior Project Manager & Scrum Master

Institution: EOH Microsoft Coastal (EOHMC)

Venue: 1st Floor Block B Granger Bay Court, Beach Road, V&A Waterfront Cape Town

Andile Koka

I would like to start this interview with the introduction of the study I am doing, I am conducting a study titled "Quality Assurance in Scrum Projects: a case of software development processes among Scrum teams in South Africa". The purpose of this study is to explore the extent to which software quality assurance measures can be understood and applied to maximize the quality of software project developed under Scrum methodology. Secondly I would like to thank you for participating in my study.

Question

What quality assurance measures/processes do you use in your team to ensure that the projects you develop under Scrum methodology meet quality standards?

EOHMC, AP – R1

Actually what we are working on at the moment but in general what happens is when we create the requirement we don't use formalized requirement or functional specification document anymore, we found that those huge documents just don't work anymore no one ever reads them and they get emailed and you lose version control and everything just gets very confusing. We use online tools at the moment for central requirements so the client can look at the small sheet which is basically an online spreadsheet which we have one per project and that's where the user stories go in and the requirements beneath those user stories so as a user I want to log into the application and then underneath that we will have a requirement so it must have a login button, it must have password button so it's all the bits and pieces there. Then something called envision is where we have our mock-ups and our process flows so before we can start the

development we've got process flows set out which takes you through what happens from a user generally sitting at the computer using the program as well as the mock-up screens because mock-ups have click-throughs so the client will then comment and sign off those requirements before we start and then any changes along the way will be happening there. From a quality assurance point of view, once the requirements are developed we follow pretty straight methodology we do fact testing, firstly the developers do their testing which is unit testing and all their side of things which go to SAFe testing which is done within the team and depending on what product…. Because we have lots of streams of work happening so depending on whether users impact on the membership system on the client side or whether it's solely on our application, it either goes to the client QA team for again functional testing or it goes to one of our client resources for user acceptance testing.

So we do Dev in local, we have FAT, then to UAT then we move to pre-production and then to production. Pre-production merge production exactly and it sits on the same database cluster, the same environment as production, UAT still sits within the development environment in terms of the server perspective, so it does not have necessary all the certificates so it's not exactly a mirror of production.

So from a quality perspective, on development, no developer may check-in without a code review and where we have code reviews, a senior can review juniors or intermediates and can review each other. Intermediates if they prove themselves could potentially review juniors or each other so no one can review up they can review side or down and that's only if the team leader is comfortable with the person doing that but we have put a check analysis in TFS so you may not check-in any item in TFS that does not have a work item associated with it, so you have to create a work item in TFS and associated it with a work item and you may not check-in if you don't have the name of the person reviewed your code.

With regards to documentation, our developers are quite big on commenting the code and to note as to what is gone in and what has changed and they also might not work on the work item they have done. So that I am not really concerned about it because we have a very collaborative process that we do when the feature get built it, gets built right the first time as opposed to rebuilding the code to bug fixes. From the requirement side we have found that side conversations have happened and then my BA will come up and say where does this requirement comes from? So we try to be quite strict about making sure that if it's a change request, we have a change request form filled in and signed by a client. If it's a requirement

change for technical reason that it happens that it gets put into a smart sheet first. In terms of not reading documents we moved away from documents because now we have a central Internet based list of requirements and part of what the developers need to do is to work against those requirements to make sure they build actually what is required and we have found that a lot better working because everything is put in these intersection on smart sheet we can see exactly what is really happening. Our main issue there is how we link it up because TFS has our work items with the requirements sitting in smart sheets so we want to see a better product out there that will integrate with TFS so that we don't have to double capture things.

From delivering on time perspective, we have a change control process so any change request that comes in needs to follow the process, basically it says what needs to be done and then why? So what is the impact if it does not happen and its impact if it does happen and then the priority from the client? The client has to sign it off and there is also…. We have a good relationship with the client so if new requirement comes in, we size it and then we will let the user know whether it is possible to fit it in with the next release or the next sprint if possible to just put it in or more likely we will have a discussion about them… if we are doing this now what is going to fall away because there is this new requirement?... In terms of deliverables that last 12 schedules, we have had, we made 10 on time with all requirements delivered and the two we didn't make it because of the issues on our side. It was an integration with the client side so there were delays from that side and also delays on business questions holding off the deployment so generally we have had a very high success rate with Scrum.

Question

From quality assurance perspective what is the significance of the quality measures mentioned above, do they help to improve quality?

EOHMC, AP – R2

Look the code reviews are good from both quality assurance perspective and also from the training perspective because that's how the guys are going to learn if they don't get told what they have done even if it might be functionally correct it doesn't not follow best practices so they need to follow that as well so certainly we see a very good improvement in the developers or in my team. Beyond that in terms of products at [redacted] we have one of the most stable products in the system, at any given time we have less bugs maybe one or two production bugs, we have incredible low number of bugs. Any bugs that come up in the production system are

generally fixed within a day or so. We have very stable platform, the issues that come back are usually not in terms of the technical implementation. So it's more change requirements coming in than the bug fixes.

User involvement (ensuring quality at design mode or dev phase) making sure you have quality objectives.

We….. there is one BA in my project which is a bit... she works very close with the client in terms of the requirements. The clients give us a brief of what they want and then most of the elaboration has to happen on our side. Technically the BA is only supposed to be a technical BA creating functional requirements but what we found is that sometimes the client hasn't necessary thought through all the business logic, the business rules side of things so we have workshops where we ask business questions like have you thought about this….?, what about that? So at least we do due diligence if the client say we are not issuing that at the moment so that's fine we have done our part. But yes both [redacted] and I are involved in the process. We make sure that we both gather requirements together and once those requirements are gathered the client have to approve them before we go ahead and do development.

From the testing perspective, for a long time testing was handled between my BA and myself but at the beginning of the year we got a senior tester to help with the functional testing of the new product, we had a very tight deadline but also to look at our processes and look at what sort of testing harnesses we can put in, in order to improve our QA processes. So some of the things that have come out of that is that the client is investigating the number of quality assurance tools of which is quality sector also beyond that looking at what can be automated from testing perspective and something we are looking at from the dev perspective even though we haven't built that yet because the business has to bring the TFS infrastructure in house first which we are planning to do it but we don't have a date to that but when that happens then, when the developer has checked in the code then we can have automated test builds get created once the test has happen then we can sign that off and then we can have automated UAT build. That will make our lives a lot quicker and easier because things will be picked up quickly.

Question

How do you ensure/measure customer satisfaction or how do you know that the customer is satisfied with the product you developed?

EOHMC, AP – R3

We ensure that as many government factors we can put in place to ensure quality is there. We have demos, the client decides before we can go into production. We work very closely with the client team to make sure that everyone is happy with the delivery.

Question

What is the success rate of the project developed under Scrum methodology?

EOHMC, AP – R4

Well, we are very successful with the Scrum methodology because we are in an environment where there is a lot of turn in terms of requirements and in terms of priorities from the client side. We would not work very well with other methodologies like waterfall method because there is so much change Scrum works best for us. It is very easy for us to lock down two weeks sprint. Locking down three months' work just won't work for us. Literally every month if not two more often if the project changes the requirements will change. We have a lot of project stakeholders who colour in and colour out, they sweeping, they say no you must do it this way they sweep out. We do have a process which says no you can't put in a new requirement that hasn't been approved by the product board so that's when we hit our backlog hopefully everything approved. We have had a lot of issues even when something has been approved but they turn around and say actually we didn't think this, you have got to do it this way instead or actually can't you just slide this as well? So Scrum and agile works better for us because it allows us deal with that sort of change in a way that waterfall methodology would not work.

Question

What are challenges you are facing with the implementation of Scrum methodology?

EOHMC, AP – R5

Not specifically with Scrum I think one of the biggest challenges is to actually track tasks properly in terms of something like burn down chart. If team members are not closing off their tasks in TFS then it is very difficult to give an accurate representation of velocity. The issues are not with the process, the issues are more with making sure that the team do due diligence in terms following the process.

Date: Friday, 12 June 2014 16:00 – 16:30

Interviewer: Andile Koka

Interviewee: [redacted] – Scrum Master/Coach

Institution: ScrumSence

Venue: 3st Floor Old Mutual, Mutual Park Pinelands

Andile Koka

I would like to start this interview with the introduction of the study I am doing, I am conducting a study titled "Quality Assurance in Scrum Projects: a case of software development processes among Scrum teams in Cape Town South Africa". The purpose of this study is to explore the extent to which software quality assurance measures can be understood and applied to maximize the quality of software project developed under Scrum methodology. Secondly I would like to thank you for participating in my study.

Question

What quality assurance measures do you use in your team to ensure that the projects you develop under Scrum methodology meet quality standards?

ScrumSence, JP – R1

Scrum does not prescribe quality assurance at all, it does not tell in any way how to do the quality assurance measures or prescribe quality assurance processes for quality assurance measure or anything like that. What it does do is…, it tries to create an environment and it enables it. What happens often is somebody needs something, somebody goes along in that specification which is different from the need and you build something. So what Scrum tries to do or what a lot of agile processes try to do is to enable those two circles to overlap more and overlap better. So it's about creating an opportunity for the right communication channels and the right people to communicate with each other, because ultimately that's what it's about.

So how does one bridge the gap created by Scrum, since Scrum does not directly address quality?

So it depends on the definition of quality right, but quality is defined as value to someone from a person, so depending on that person, the definition of quality is going to change. So for

developers, quality might be something that is building a product very fast or retrieving data very fast, from users, quality is something that is really easy to use, from product owners, quality might be something that gets shipped very quickly, so for different people quality means different things. So you need to start answering those questions very effectively. So between the development team and the testing team, what you need to do is try and organize or find a way who is the person in the quality is value to person so who is the person? And then what does that mean to them? So what does quality mean to them? And then you can come with a strategy once you have answered those questions. It is not important to come up with a strategy if you don't understand what it is that you are building it for because then I can put a lot of processes in place, I can follow those processes and I am still not really stopping the problem. So to bridge those gaps what you need to do is…, you need to have a right conversation with the right people. So you need testers who are speaking to developers and testers who are speaking to business people. First of all understand the need of the business or the need of the person as well as understanding what the developers are going to be building. From there you can then understand where those gaps are, where people are missing each other? So maybe there is a gap that's between the need and the specification, maybe there is a gap that's between the need and what was built and all of those things are far more about the conversations that people have and physical actual processes. So the way that Scrum encouraging that is…, testers are part of Scrum team so testers and developers work together. Another way of doing is that you test the whole time so you are not waiting until the very end of the project before you start testing that project. You are testing each and everything that the developers do and each and everything that's gets shipped so you can start to find issues quicker and the another thing that you are doing is that you build the smaller pieces so when you develop stuff you develop it in smaller chunks and by developing things in smaller chunks it is much easier to understand first of all what is going wrong, second of all what the differences are between the last chunk you shipped and the one you busy with. So on that point of view, those are the kinds of things that Scrum encourages. It encourages daily stand-ups, it encourages synchronize what they are doing each day. So having a conversation about I am testing this and I tested this yesterday and it didn't work so the developer and tester can sit together and figure out what went wrong and fix it here and there it does not wait for another forty five week iteration or another whole development circle before the developer gets back to it. It is much easier to fix if the context is still fairly new in your head.

Unit testing, there is lot of debates that are going around in the agile community because they have their place like everything so everything has got a place and point where it's going to be useful so everything is applicable within a specific context. So unit testing can be very useful and can be very hobble to keep a code base clean, it can also waste a lot of time if you are testing the wrong stuff so you don't want to write unit tests just for the sake of writing unit tests because somebody said it was a good idea you need to apply your mind, you need to make sure you are writing the unit test for the product and are going to benefit from those unit tests later on. Because in the beginning you have got a small code base about a thousand lines of code or whatever, it is really small. It is very easy to debug, it is very easy to add things into that code base and to take things away from that code base. In the beginning of projects people are oh…. well our code base is fine, it's small enough we don't need to do that. What happens in the end though is that code base becomes massive, five years, ten years and fifteen years down the line whatever it is and twenty five other developers were there, they have contributed and now all of a sudden your code base is fifteen to twenty thousand billion lines of code or whatever it is, it's much harder in those instances to find paths. In those instances it's very beneficial to unit tests like I said though, you must unit test at the right time and at the right level of granularity.

On the other hand, code reviews are awesome, code reviews are very cool way, to first of all get developers to learn how each other codes.

Question

From quality assurance perspective what is the significance of the quality measures mentioned above, do they help to improve quality?

ScrumSence, JP – R2

So what really is important is that you are finding things quicker, so if you find something fairly early then you can change it before it becomes a problem. Also you want to make sure that you find problems as early as possible because the earlier you find them, the easier you first of all fix them and the lesser of a problem they become so that's one of the significance.

To speak to some of the other stuff we were talking about, the things like **how do you ensure that the requirements are met and whatever you building is doing the right thing**? What agile methodologies do is, they try to close those feedback loops really quickly so every release or every sprint or every time there is a shippable increment then that shippable increment is

presented to the customer or the user or whoever the stakeholders is so that they can see exactly what it is and then recommend changes. So ninety percent of the time what happens is that, people do not know how to put it into words accurately what it is that they really need and more often than not once they have seen it they realize that there are things that are wrong and they want to make changes and that's just the nature of reality because sometimes a business person does not understand the abilities of the technologies themselves so it is very hard for them to describe and explain exactly what it is they need until they have seen a portion of it, if that make sense. So it's much easier for them to see something then understand oh actually we needed that button in a different place it really has to go from point A to point B and what we need is to have all that information in one place instead. So those kinds of changes, so that's what agile is encouraging. It encourages ways to close those feedback loops as early and as often as possible because the longer you wait to close the feedback loops the more difficult it is for you go and change the system.

Question

How do you enforce SQA measures to ensure that everyone adheres to standards set in place?

ScrumSence, JP – R3

Scrum encourages self-organization, it encourages teams to self-organize so a Scrum master is in charge of the process and helps to keep the team on track but most and the rest of the stuff like code reviews and all of that comes from within the team itself. My experience is that people have their own ideas about how to do things, the abilities to make the right decisions. They are far more likely to buy into it and carry it out so they hold themselves and each other accountable to those things as well as set apart in their craft in what they are doing if they are excited about what they are building and they want to make sure it is of high quality then they will hold themselves to those standards.

Question

What is the success rate of the project developed under Scrum methodology?

ScrumSence, JP – R4

To be honest I do not have statistics of that at all so I would invite you to Google and ask people and have a look at what you can find on Scrum alliance and agile alliance see what people are saying. More often than not it is not the project that fails but it is the agile implementation that

fails so agile is about creating better places to work on so that does not require people to revert back to working overtime, killing themselves, trying to finish the project by and in date which more often than not they do successfully at the risk and expense of all quality. So the dates affect the quality.

Question

How do you ensure/measure customer satisfaction or how do you know that the customer is satisfied with the product you developed?

Question

What are challenges you are facing with the implementation of Scrum methodology?

ScrumSence, JP – R5

The challenges facing Scrum is that it's difficult to change the way you do things at the moment so more often than not, changing the way that you work is not easy in all organizations. They do not have openness, they do not have transparency, they do not have the trust structures in place and it is very difficult to change the way you work. The communication structures are very complicated and Scrum agile work better if communication structures are not complicated.

Annexure 2: Research Ethics Letter

Cape Peninsula University of Technology

Introductory letter for the collection of research data

Andile Isaac Koka is registered for the M Tech (IT) degree at CPUT (213304376). The thesis is titled Software quality assurance in Scrum projects: a case study of development processes among Scrum teams in South Africa, and aims to explore the extent to which software quality assurance processes can be understood and applied to maximize the quality of software in Scrum projects. The aim is to identify five Scrum projects with different teams and operating circumstances, so as to understand common quality assurance processes upon which SQA inferences on Scrum can be made. The goal is to contribute additional insight on quality assurance in Scrum, so as to contribute towards improved quality assurance practices in agile methods, generally. The supervisor(s) for this research is/are:

Prof Nhlanhla Mlitwa

Email: MlitwaN@cput.ac.za

Tell: 021 464 7201

In order to meet the requirements of the university's Higher Degrees Committee (HDC) the student must get consent to collect data from organisations which they have identified as potential sources of data. In this case the student will use Interviews and Direct-Observation to gather data.

If you agree to this, you are requested to complete the attached form (an electronic version will be made available to you if you so desire) and print it on your organisation's letterhead.

For further clarification on this matter please contact either the supervisor(s) identified above, or the Faculty Research Ethics Committee secretary (Ms. V Naidoo) at 021 469 1012 or naidoove@cput.ac.za.

Yours sincerely

Nhlanhla Mlitwa

22 April 2014

<<On company letterhead>>

I <<*insert name*>>, in my capacity as <<*insert position in company*) at <<*insert company name*>> give consent in principle to allow <<*insert student name*>>, a student at the Cape Peninsula University of Technology, to collect data in this company as part of his/her M Tech (IT) research. The student has explained to me the nature of his/her research and the nature of the data to be collected.

This consent in no way commits any individual staff member to participate in the research, and it is expected that the student will get explicit consent from any participants. I reserve the right to withdraw this permission at some future time.

In addition, the company's name may or may not be used as indicated below. (Tick as appropriate).

	Thesis	**Conference paper**	**Journal article**	**Research poster**
Yes				
No				

_____ _____

<<Insert name>> *<<insert date>>*

Annexure 3: Interview Request Letter/s

• PO Box 77000 • Nelson Mandela Metropolitan University
• Port Elizabeth • 6031 • South Africa • www.nmmu.ac.za

Nelson Mandela Metropolitan University
for tomorrow

North Campus
Faculty of ICT
Tel . +27 (0)41 504 3314 Fax. +27 (0)504 3313
andrew.rutherford@nmmu.ac.za

17 April 2014

To Mr Andile Koka
Cape Peninsula University of Technology
Cape Town

Re: Software Quality Assurance in Scrum Projects: A Case Study of Development Processes among Scrum Teams in Cape Town, South Africa

Dear Mr Koka

I refer to your letter dated 7/03/2014 regarding permission to conduct an interview with me as part of on-going research for the above-mentioned subject.

I am pleased to inform you that your request has been approved. You may contact me with your preferred date and time to confirm availability.

Yours Faithfully

Andrew Rutherford
Senior Lecturer
School of ICT
Faculty of Engineering, Built Environment and Information Technology
Nelson Mandela Metropolitan University
Phone : +27 41 504-3314
Fax : +27 41 504-3313
Andrew.Rutherford@nmmu.ac.za

TRUWORTHS

3 July 2014

IT Department,
Cape Peninsula University of Technology

To whom this may Concern,

This letter serves to confirm that Andile Koka who is completing his Masters Degree at Cape Peninsula University of Technology had our authorization to interview some of our IT Developers.

Andile, interviewed our Developers on the 26 May 2014.
He interviewed our Scrum Master, Lead Developer and observed a Stand-up session.

Regards
Rene Roos
IT Store Operations Systems Manager

Annexure 4: Interview Questions

Question

What quality assurance measures/processes do you use in your team to ensure that the projects you develop under Scrum methodology meet quality standards?

Question

From quality assurance perspective what is the significance of the quality measures mentioned above, do they help to improve quality?

Question

How do you ensure/measure customer satisfaction or how do you know that the customer is satisfied with the product you developed?

Question

What is the success rate of the project developed under Scrum methodology?

Question

What are challenges you are facing with the implementation of Scrum methodology?

Annexure 5: The Agile Scrum process

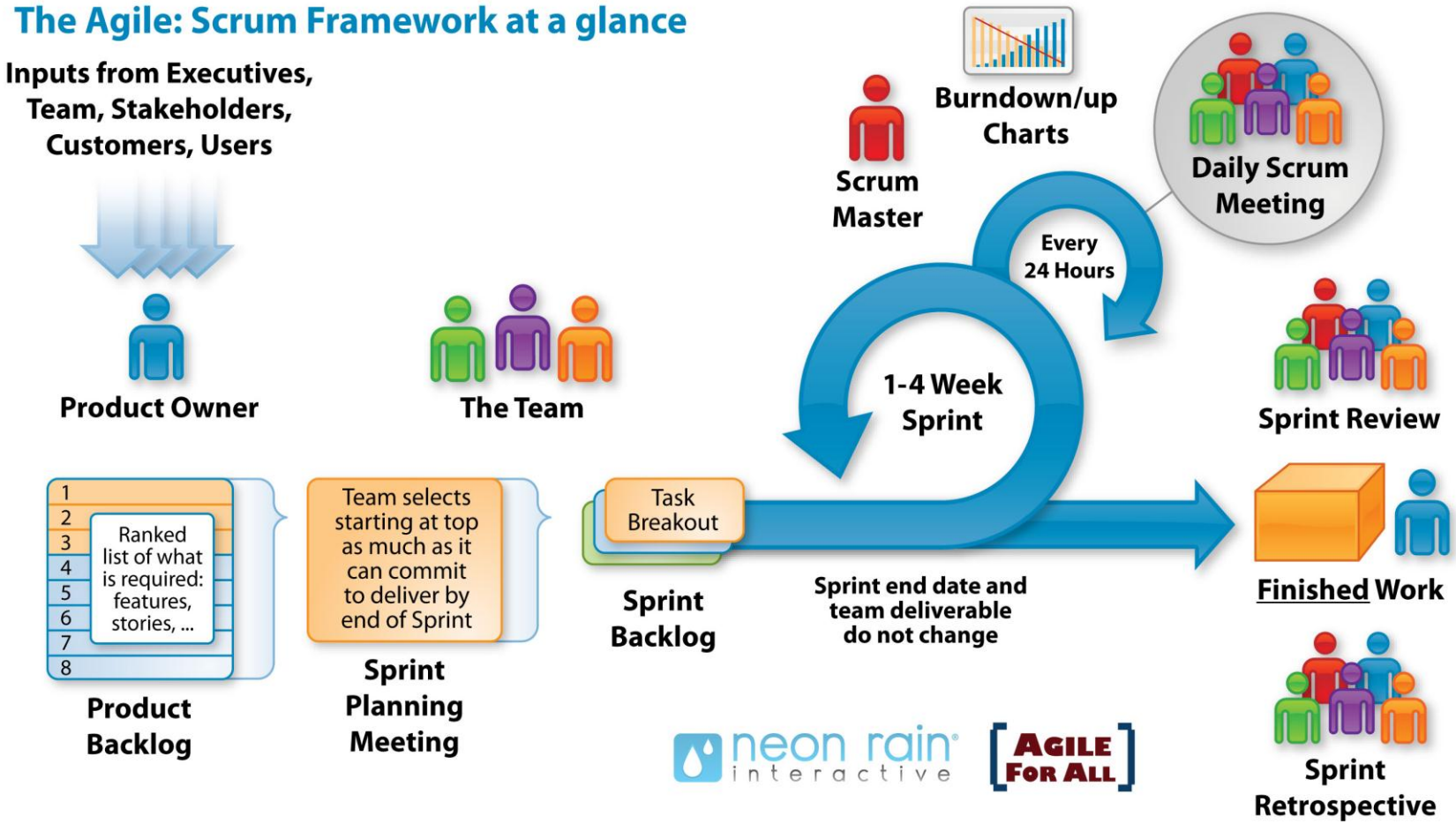

Figure 4: The Agile Scrum Process Source: SoftWaysSolutions, 2012