Software Quality Assurance in Scrum

The need for concrete guidance on SQA strategies in meeting user expectations

A Thesis Presented to the
Department of Information Systems at the University of Cape Town
In Partial fulfilment of the requirements for the
Masters in Information Systems (INF5005W)

BY:

Tiisetso Khalane
KHLTII001

February 2013

DECLARATION

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.

2. I have used the APA convention for citation and referencing. Each contribution to, and quotation in, this Masters Thesis, from the work(s) of other people has been attributed, and has been cited and referenced.

3. This Masters Thesis ***Software Quality Assurance in Scrum*** is my own work.

4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

5. I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.

SIGNATURE: Tiisetso Khalane

DATE: FEBRUARY 2013

ACKNOWLEDGEMENTS

I would like to pass my sincere gratitude to my supervisor Ms. Maureen Tanner for her guidance during the course of my Masters Course. I would also like to thank Prof. Irwin Brown for his advice and recommendations.

I wish to thank the staff and students in the Department of Information Systems at the University of Cape Town, particularly the Masters class of 2011.

My gratitude also goes to my girlfriend, Mots'elisi Pearl Mokhachane for her love and patience with me, for giving me the encouragement that I needed throughout the course, and for helping me overcome some of the difficulties I faced.

To my friend Moletsane Monyake, for challenging my ideas, my thinking, and my understanding of Scientific research, thank you.

I wish to pass my deepest gratitude to my colleagues and friends who contributed immensely and gave me countless ideas on my research topic.

Most importantly, my sincere gratitude goes to Tom O'Rielly for the vested interest he had on my thesis, for opening many doors for me, and for giving me direction when I most needed it.

May God bless you!

ABSTRACT

The purpose of this study is to identify and present the concerns of project stakeholders in relation to Software Quality Assurance (SQA) in a Scrum environment. Guided by the tenets of Classic Grounded Theory Methodology, this exploratory and inductive case study presents a broad range of SQA concepts related to the main concern of "*Meeting User Expectations*". In trying to resolve the main concern, the Scrum project stakeholders alluded to lack of "*Concrete Guidance*" on SQA strategies, tools, and techniques in Scrum. The lack of concrete guidance in Scrum requires a development team to devise *"Innovations"* which may include *"Adopting Practices"* from other methodologies and carefully designing the *"Process Structure"* to accommodate the *"Adopted Practices"*, ensure *"Continuous Improvement*" of the process, and provide an environment for "*Collaborative Ownership"*.

In addition to the "*Need for Concrete Guidance*", the study reveals two other important concepts necessary for *"Meeting User Expectations":* the *"Need for Solid User Representation*" and the *"Need for Dedicated Testing*". While some Agile proponents claim that the Agile SQA practices are adequate on their own, the study reveals a number of challenges that impact on a team's ability to meet user expectations when there is no dedicated tester in a Scrum environment. The challenges include increased *"Capacity Demands"*, *"Testing and Quality Issues"*, and *"Lack of Testing Expertise"*.

By demonstrating the incompleteness of Agile methods with particular attention to the lack of concrete guidance in Scrum, the study draws on method tailoring literature to argue for customisation of Scrum. The study further proposes that Scrum needs to be viewed as a framework of 'empty buckets' which need to be filled with situation specific SQA practices and processes in order to meet user expectations.

DEDICATION

I dedicate this thesis to the most precious person in my life – my mother, 'Me' '***Matiisetso Khalane***. With this thesis I pledge to pursue the promise - the dream, and to honour your teachings; so that you may see in me, what you would have become if you had the opportunities that I have.

LIST OF TABLES

Table 2.1 Agile Manifesto 27
Table 2.2 the Scrum framework 29
Table 2.4 Scrum roles 31
Table 2.5 Scrum Ceremonies 32
Table 2.6 Scrum Artefacts 33
Table 4.1 Team Composition 46
Table 5.1 List of Participants 53
Table 5.2 Concept-Indicator model and emergence of Capacity Constraints 56
Table 7.2 Dependency resolution 102
Table 7.3 Traditional SQA vs. Agile SQA 108

LIST OF FIGURES

Figure 1-1 Systematic reviews of literature on Agile 14
Figure 2-1 Scrum methodology (Schwaber, 1995, p. 10) 30
Figure 5-1 The Research Process: Applying Classic GTM 49
Figure 5-2 Concept indicator model comparing subsequent incidents with concept (Adolph et al., 2011) 55
Figure 5-3 Constant comparisons for Capacity Constraints 57
Figure 6-1 Aspects of Software Quality Assurance in a Scrum environment 68

TABLE OF CONTENTS

Declaration i

Acknowledgements ii

Abstract iii

Dedication iv

List of Tables v

List of Figures vi

Chapter 1. Introduction and Background to the study 11

1.1 Introduction 11

1.2 Background 11

1.3 The State of Research on Scrum and Software Quality Assurance 14

1.4 Research Objectives and methodology 16

1.5 Research Context 17

1.6 Organisation of Paper 18

Chapter 2. Literature Review 19

2.1 Software Quality 19

2.2 Traditional Approaches to Software Quality 21

2.2.1 Software Quality Assurance 21

2.2.2 Software Quality Evaluation 22

2.2.3 Software Quality Management 22

2.3 Software Testing 24

2.4 Agile Methodologies 26

2.5 Scrum Methodology 28

2.5.1 Scrum 28

2.5.2 Scrum Workflow 29

2.5.3 Scrum Roles 31

2.5.4 Scrum Ceremonies 31

2.5.5 Scrum Artefacts 32

2.6 Disparate Field Studies on Scrum 33
2.6.1 Overview 33
2.6.2 Studies on Enablers of Quality 33
2.6.3 Studies on Challenges to Achieving Quality 34
Chapter 3. Research Methodology 36
3.1 Methodology 36
3.2 The Grounded Theory Method 36
3.2.1 Glaserian versus Straussian GTM 37
3.2.2 Classic GTM 38
3.3 Research Paradigm 40
3.4 The Case Study Strategy 40
3.5 Reliability and Validity in Case Research 41
3.6 Qualitative versus Quantitative Data 42
Chapter 4. Context Description 43
4.1 Case Organisation 43
4.2 Project Background 44
4.3 The Project Team 45
4.4 The History of Scrum at SAIT 46
4.5 Scrum Implementation by the Portal Project Team 47
Chapter 5. Research Approach 48
5.1 Data Collection 49
5.2 Canons of Classic GTM 54
5.2.1 Data Incidents, Concepts, and Categories 54
5.2.2 Constant Comparative Method and Rigour in Classic GTM 54
5.2.3 Theoretical Sampling 58
5.2.4 Theoretical Memoing 59
5.3 Data Analysis 60
5.3.1 Open Coding 61
5.3.2 Selective Coding 61
5.3.3 Theoretical Coding 63
5.4 Scaling Up 64

5.4.1 Theoretical Saturation 64
5.4.2 Delimiting the theory 64
5.4.3 Sorting 65
Chapter 6. Findings 67
6.1 Meeting user expectations 67
6.2 Concrete Guidance 69
6.2.1 Need for Concrete Guidance 69
6.2.2 Process Structure 71
6.2.3 Adopted Practices 75
6.2.4 Guiding Principles 79
6.3 Dedicated Testing 82
6.3.1 The Need for (or lack of) Dedicated Testing 82
6.3.2 Challenges due to the absence of dedicated testers 83
6.4 Business Buy-in 87
6.5 Solid User Representation 89
Chapter 7. Discussion 91
7.1 Method tailoring 93
7.2 Process Structure 96
7.2.1 Adopted Practices 96
7.2.2 Workflow Design 98
7.2.3 Developer testing 98
7.2.4 Collective Code Ownership 99
7.3 Work Cordination and Dependency Resolution 100
7.4 Dedicated Testing 103
7.4.1 Consequences of the Lack of Dedicated Testing in Scrum 104
7.5 Business buy in 105
7.6 Software Quality Assurance and Agility 107
7.7 Empirical Process Control 109
7.8 Implications for Practice 110
Chapter 8. Conclusion 111
8.1 Research Objectives 111

8.2 Summary of findings 111
8.3 Future research 113
8.4 Research Contribution 114
8.5 Limitations 114
Chapter 9. References 115
Chapter 10. Appendices 122
10.1 Appendix A: Interview Questions 122
10.2 Appendix B: Concept-Indicators (Coding Sheet) 123
10.3 Appendix C: Sample Memo 147
10.4 Appendix D: Real Time Memoing and Theorisation 149

CHAPTER 1. INTRODUCTION AND BACKGROUND TO THE STUDY

1.1 INTRODUCTION

The purpose of this chapter is to introduce the study by providing an introductory background, research gaps, research objectives, and an introductory section on the research methodology. The chapter has five sections. The first section provides a background to the study by detailing the need for organisations to pay attention to software quality. It also provides recent adoption of Agile software development methodologies such as Scrum as an alternative for overcoming some of the downfalls of traditional plan-based software development methodologies. The second section provides detailed research gaps regarding Scrum and Software Quality Assurance (SQA). The third section provides the research objectives based on the identified gaps and introduces the Grounded Theory Methodology. The fourth section gives a brief introduction to the case organisation – SAIT – and the Portal Project. The fifth section provides the layout and organisation of this research document.

1.2 BACKGROUND

As Osterweil et al. (1996) predicted, the ability to deliver successful Information Technology (IT) projects has become a critical and strategic necessity for contemporary organisations (Owens & Khazanchi, 2009). An average company invests five to ten percent of revenue on IT initiatives (Charette, 2005). A large amount of the spending goes to new software development projects aimed at improving the future prospects of the organisation.

IT initiatives in general continue to face high cost overruns, scope creep, cancellation, quality issues, and customer complaints (Cicmil, Hodgson, Lindgren, & Packendorff, 2009; Smyth & Morris, 2007). A failing software project can jeopardize an organisation's competitive position and can have far reaching consequences (Charette, 2005). Software quality is one critical component of the criteria used to measure success of a software development project. Quality is a multifaceted and a complex concept (Kitchenham & Pfleeger, 1996). This has lead to many definitions of quality as stipulated in the Chapter 2 of this document. For the purposes of this

study quality refers to the ability of a software project to meet business requirements and add value to the user. Although much effort has been put on identifying ways of ensuring software quality, software projects continue to fail (Charette, 2005).

Traditional software development methodologies such as the Waterfall model are said to have contributed largely to the high failure rates of software projects (Kayes, 2011). A 2006 report by the Standish Group, (as cited in Kayes (2011)) indicates that 41% of agile projects succeeded as compared to 16% of Waterfall projects. Traditional software development methodologies are characterised by formalised and rigid structures for all types and sizes of projects (Huo, Verner, Zhu, & Babar, 2004). They do not allow flexibility of incorporating late changes to requirements (Huo et al., 2004). These traditional methodologies are also known for their big design upfront approach to software development. With this approach, significant modelling and documentation occur before any development activities. This has proven risky in practice as it is often the case that users want to add or change some items in the requirements specification later in the project thus leading to cancelations, scope creep, incorrect software behaviour, and other software quality problems (Ambler, 2005).

Agile software development has emerged as an alternative to organising and managing complex project undertakings by providing mechanisms to adapt to constant project change (Strode, Huff, Hope, & Link, 2012). Agile software development is defined as "the continual readiness of an Information Systems Development (ISD) method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment" (Conboy, 2009, p. 340). The aim of Agile methods is to minimise the cost of inevitable change (Highsmith & Cockburn, 2001). Timperi (2004, p. 1) states that the "aim of Agile methodologies is to deliver business value rapidly by delivering working software frequently".

Agility and software quality are becoming more important due to the pace of technology change and market dynamics (Winter et al., 2008). Traditional quality approaches were tightly coupled with traditional software development methods such as the Waterfall model. These

quality approaches focused on harnessing complete, testable, and consistent requirements, traceability to design, code and test cases, and heavyweight documentation (Winter et al., 2008, p.1). On the other hand, agility redefines software quality assurance and management practices and makes some traditional quality responsibilities and roles such as Quality Assurance Engineer less valuable (Talby, Keren, Hazzan, & Dubinsky, 2006; Sfetsos & Stamelos, 2010). The Agile Manifesto has become the blue-print for adoption of these light-weight methodologies (Fowler & Highsmith, 2001).

The Scrum Methodology is one Agile approach widely followed in industry (Moe & Dingsøyr, 2008). Although Scrum is gaining popularity, few empirical studies have investigated how organisations can achieve quality requirements in Scrum projects. Beyond the case studies by Schwaber (2004) little is known about how Scrum teams achieve software quality requirements. In recent years, Scrum has become one of the most commonly used project management methodologies in practice (Kayes, 2011). While the prevalence of Scrum in industry justifies attention from the research community, there are significant gaps in the academic literature on Scrum (Sfetsos & Stamelos, 2010) and software quality in particular (Timperi, 2004). Scrum and Software Quality Assurance are the focal areas for this study.

Systematic reviews of empirical literature on Agile software development by (Dingsøyr, Dybå, & Abrahamsson, 2008) and (Dingsøyr & Dyba, 2008) revealed that only 21% of all the research papers under the review were published in scientific journals. Almost half of these 21% were published in a magazine, namely the IEEE Software, which is not a primary academic journal. . Figure 1-1 depicts the results of the review.

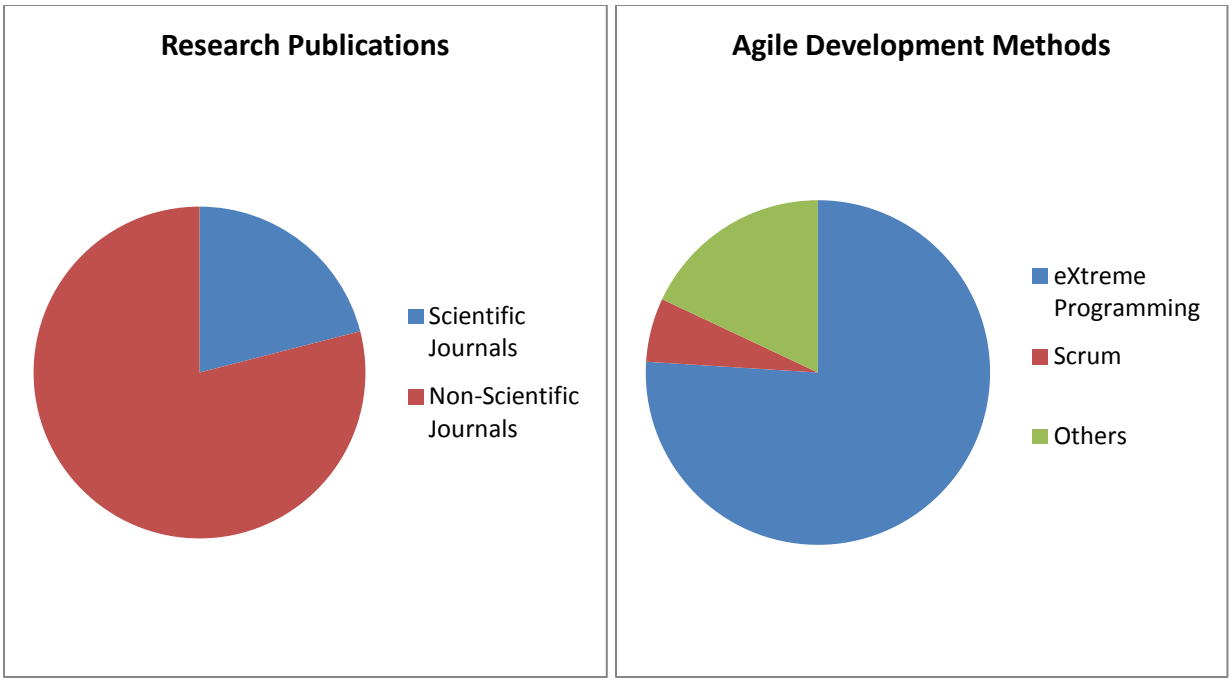


Figure 1-1 Systematic reviews of literature on Agile

Further, 76% of all the research papers under review focused on eXtreme Programming (XP). Another review, as cited in (Dingsøyr et al., 2008) corroborated these findings and discovered that only 6% of published studies focused on Scrum and another 6% focused on software quality assurance. Of a few studies reviewed by (Bhasin, 2012), all focused on XP and none on Scrum. Thus Dingsøyr et al., (2012) bemoan the absence of a unified framework that lays down fundamental principles and provides a coherent structure to disparate streams of research work.

1.3 THE STATE OF RESEARCH ON SCRUM AND SOFTWARE QUALITY ASSURANCE

As illustrated in the preceding section, most studies in the Agile literature have investigated XP. "Other management-oriented approaches, such as Scrum, are clearly the most under-researched compared to their popularity in industry" (Dingsøyr & Dyba, 2008, p. 852). As per the review by Dingsøyr and Dyba (2008), Scrum was studied in only one empirical research article each.

Dingsøyr, Nerur, Balijepally, and Moe (2012) provide a summary of five Special Issues from major Journals on Agile methods between 2003 and 2011. The Special Issues addressed a

range of topics including adoption, distributed environments, flexibility, and control. A review of the articles within the Special Issues reveals that none of these Special Issues or articles published within them focused explicitly on how software quality may be achieved when using Agile methods. Agile research tends to treat quality as a bi-product of a range of aspects treated in isolation by the research community. A majority of articles that have touched on software quality and software quality assurance investigated XP not Scrum. Some discussed quality from the broader Agile umbrella and although the Agile methods share similarities in underlying principles, each has its own practices that make it unique and therefore warrant separate treatment.

The focus on XP is more prominent in the European Journal of Information Systems Special Issue on Agile methods published in 2009 (Abrahamsson, Conboy, & Wang, 2009). None of the articles (Maruping, Zhang, & Venkatesh, 2009; Mangalaraj, Mahapatra, & Nerur, 2009; Cao, Mohan, Xu, & Ramesh, 2009) in the Special Issue focused on Scrum. For example, Maruping, Zhang, et al. (2009) focused on 56 XP projects and discovered that collective ownership and coding standards contribute to improved technical quality.

Another Special Issue on Agile methods in Information Systems Research publication (Ågerfalk, Fitzgerald, & Slaughter, 2009), looked at flexibility in distributed systems development. The central theme in Special Issue was agility. Most articles discussed agility from a XP perspective while some discussed the broad Agile methods umbrella. A study by (Maruping, Venkatesh, & Agarwal, 2009) uses control theory to argue that the relationship between software quality and use of Agile methods is dependent on requirements change and project governance modes. Although this study makes a significant contribution, it too focused on XP. It is important to highlight that although there is a disproportionate attention afforded to the methods, Conboy (2009) makes a stellar contribution in this special issue by clarifying the definition of agility, flexibility, and leanness.

Although Scrum and XP share some similarities such as flexibility, short delivery cycles, and simplicity (Maruping, Venkatesh, et al., 2009), there are differences in that “Scrum targets the planning and management of development projects whereas XP concentrates on supporting technical implementation steps” (Overhage & Schlauderer, 2012, p. 5453). It is important for

researchers to treat these two methodologies separately in order to improve understanding of how each works in practice.

Dingsøyr and Dyba (2008) believe that theory and research in some Agile methodologies such as Scrum is still nascent, and this calls for more qualitative studies. According to Dingsøyr, Dybå, and Abrahamsson (2008, p. 88), "nascent research areas are characterized by open-ended inquiry about the phenomenon which can generate suggestive theories". The researcher therefore embarked on an exploratory and inductive theory building case study focusing on Scrum and SQA as stated in the Research Objectives section as follows.

1.4 RESEARCH OBJECTIVES AND METHODOLOGY

Based on the stated gaps detailed in the preceding section, the objectives of this study were as follows:

- To illuminate the concerns of Scrum project stakeholders in relation to SQA in a Scrum environment.

- To identify and illuminate aspects of SQA in a Scrum environment

This study viewed SQA as a broad range of activities, processes, and techniques employed in Agile teams to achieve software quality requirements. Based on these objectives and the lack of inductive studies in Scrum and SQA, the researcher decided to use the Grounded Theory Method (GTM). While SQA is not a new area, its applicability to Scrum has not been given adequate attention from the IS research community as demonstrated in the literature review. According to Tan (2010) GTM is suitable for situations where the researcher aims to seek insights into a new field. GTM is particularly suitable for exploring areas that have not been explored in detail before Hoda, Noble, and Marshall (2012) and SQA in Scrum is one such area. Further, a SQA perspective on Scrum is a socio-technical territory and this makes GTM a suitable approach for generating empirical knowledge about this phenomenon (Fernández, 2004; Tan, 2010). Finally, GTM is useful for investigating IS phenomena and creating context-based, process-oriented descriptions and explanations as stated by Myers in Urquhart, Lehmann, and Myers (2010).

The support for GTM as a suitable methodology for exploring how software project stakeholders collaborate in development projects was also emphasised by Hoda et al. (2012). A GTM study must answer the question: "what is going on there and how" (Tan, 2010, p. 94) and provide a conceptualization of the concepts. According Lehmann (2010) there is a scarcity of theories and skills to generate them in the Information Systems (IS) discipline. "Substantive theories apply to the substantive area of enquiry but are independent of and beyond the data analysed and the incidents observed" (Urquhart, Lehmann, & Myers, 2010, p. 364). Lehmann (2010) suggests that building theories through qualitative and inductive approach is a well suited approach for areas where there is scarcity of theories. Gregor (2006) outlines a taxonomy of theory in IS and Urquhart et al. (2010) suggest that GTM is capable of creating theory that suits all categories of theories in Gregor's (2006) taxonomy because a theory created through GTM has constructs in the form of categories tied together by relationships.

1.5 RESEARCH CONTEXT

Researching IS phenomena often requires one to focus on wide interactions between people and technology (Lehmann, 2010). The researcher employed a case study research strategy because it is an ideal method for generating theory by learning from practitioners (Benbasat, Goldstein, & Mead, 1987). The case study methodology is appropriate for studying the nature and complexity of processes in emerging areas (Fernández, 2004). A SQA perspective on Scrum does not have a strong empirical and theoretical base hence the area can be fruitfully studied through the case study strategy (Benbasat et al., 1987).

The study investigated a Scrum team in one South African IT and business consulting company called SAIT* (not real name). The data collection lasted for 10 months from December 2011 to November 2012. The team had been using Scrum from July 2010. The project under investigation is called the Portal Project* (not real name). The researcher is a full time employee in the organisation but is not a member of the Portal Project team.

The case site was selected for both opportunistic and theoretical reasons. The one opportunistic reason is that the researcher is a software developer in the organisation and this allowed the researcher easy access to participants. The fact that the researcher is a full time

employee also offered theoretical opportunities for follow up interviews during the iterative data collection and analysis as was dictated by the emerging theory. Further observations, participation in software development meetings, and longer immersion in the field also helped theoretically in the saturation of concepts. The relatively 'young' Scrum implementation offered theoretical opportunities for investigating aspects of SQA in an environment that has not been exposed to too much customisation. Finally, the organisation has a strong experience in process oriented methodologies such as the Waterfall Model. The experience in Waterfall provided participants a better chance of judging comparatively their new concerns that resulted from the move to Scrum. The selection of the 'Portal Project' project over other teams was judgmental because that is the one project team that had close to two years of experience with Scrum and offered opportunities for investigating the aspects of SQA in a team that has just started using Scrum.

Details of the organisation, project team composition, and the implementation of Scrum are discussed in the Context Description chapter.

1.6 ORGANISATION OF PAPER

The next chapter provides a brief summary of the literature in software quality, software quality assurance, evaluation, and management. It also provides a brief overview of Agile methodologies. The chapter culminates with a review of literature in Scrum. Following the literature review chapter is a chapter on research methodology that presents methodological assumptions which guided the study. After the research methodology chapter follows a context description chapter and a research approach chapter. The research approach chapter provides a detailed 'as-lived' account of how the research process unfolded and how the classic GTM techniques were applied. Following the research approach chapter is the findings chapter which presents the results of data analysis. A discussion chapter which presents an interweaving of the findings and the literature follows the findings chapter. The conclusion chapter is the last chapter and discusses the findings in relation to the research objectives.

CHAPTER 2. LITERATURE REVIEW

This chapter is structured as follows: from Section 2.1 to Section 2.4 the focus is on a descriptive background and overview of traditional approaches to software quality management, software quality assurance, software quality evaluation, and testing. The sections are preceded by a definition of, and different perspectives on, software quality. The aim of the sections is to lay a summarised foundation, and background on traditional ways of achieving quality in software development.

Section 5 and Section 6 present a descriptive overview of software development and the Scrum methodology. These sections also aim at presenting basic definitions and summarised principles of agility and Scrum to facilitate discussion in later chapters. Section 7 presents the state of research on Scrum while Section 8 focuses on disparate research streams that have been pursued by various authors on Scrum with a particular focus on aspects of how software quality may be achieved through a Scrum process.

The next section discusses the notion of software quality and the different meanings attached to it.

2.1 SOFTWARE QUALITY

The study focuses on SQA in a Scrum environment and it is therefore fitting to briefly describe software quality. Two definitions of quality by the ISO 8402 and the IEEE respectively, are stated below:

"Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs." (Bevan, 1999, p. 89).

"Quality is the degree to which a system, component, or process meets specified requirements and customer/user needs or expectations." (Sfetsos & Stamelos, 2010, p. 44)

These definitions are stated here for reference purposes to illustrate the fact that quality is a multifaceted concept. In addition to the above definitions of quality, different people describe quality from five different perspectives (Kitchenham & Pfleeger, 1996, p. 13; Ward & Venkataraman, 1999, p. 1). These perspectives influence the process of measuring quality and a brief overview is given as follows:

"The transcendental view sees quality as something that can be recognized but not defined."

"The user view sees quality as fitness for purpose."

"The manufacturing view sees quality as conformance to specification."

"The product view sees quality as tied to inherent characteristics of the product."

"The value-based view sees quality as dependent on the amount a customer is willing to pay for it."

(Kitchenham & Pfleeger, 1996, p. 13; Ward & Venkataraman, 1999, p. 1).

Although the are many views on quality, it is paramount that organisations approach quality and quality assurance in a manner which reflects their business goals (Kitchenham & Pfleeger, 1996). Alsultanny and Wohaishi (2009) argue that software development projects should define their own explicit meaning of quality.

Osterweil et al. (1996) state that the level of quality achieved is dependent on the quality of assessment and assurance processes. The rigour of these processes depends on the type and purpose of the software product. For example, mission critical systems have high quality requirements which means that the processes must be stringent (Feldman, 2005). At a fundamental level, the processes aim at minimising deviation between the specified, intended behaviour of the product and the actual developed behaviour of the software product (Osterweil et al., 1996). The next section discusses software quality assurance, software quality management, and software quality evaluation.

2.2 TRADITIONAL APPROACHES TO SOFTWARE QUALITY

This section presents an overview of traditional approaches to achieving software quality and is presented here for reference purposes only. These approaches are: Software Quality Assurance, Software Quality Evaluation, and Software Quality Management. The overview serves as a historical background and precursor to the current agile driven approaches to quality.

2.2.1 Software Quality Assurance

SQA is one of the most important components of a software development process. As with software quality, various definitions of SQA exist in the literature. Some of the definitions in the literature are as follows:

"Software Quality Assurance is a process for providing adequate assurance that the software products and processes in the product life cycle conform to their specific requirements and adhere to established plans" (Feldman, 2005, p. 27).

"SQA is a well defined, repeatable process that is integrated with project management and the software development lifecycle to review internal control mechanisms and assure adherence to software standards and procedures." (Owens & Khazanchi, 2009, p. 245).

"Software Quality Assurance (SQA) is referred to as the activities for independent assurance of adherence to defined processes as stated in the CMM key process area on SQA" (Runeson & Isacsson, 1998, p. 685)

The shared commonality between all the definitions is assuring conformance and adherence to defined processes, standards, and procedures. The goals of SQA as stipulated by the Capability Maturity Model Integration (CMMI) include monitoring, ensuring adherence to standards and procedures, and identifying areas of improvement (Runeson & Isacsson, 1998). In CMMI, the role of SQA is to engage in monitoring of project activities using reviews, audits and measurements. SQA functions require collaboration between software development and SQA teams. The development team is responsible for implementation of the technical work (Owens

& Khazanchi, 2009, 250). The role of a SQA team is to verify a systematic execution of the rules governing unit tests, code walkthroughs, and peer reviews.

SQA includes software testing which is treated separately in Section 2.3 because it is a broad topic and is dominant in traditional (process-oriented) approaches to quality as well as Agile approaches.

2.2.2 Software Quality Evaluation

Software quality standards prescribe guidelines for measuring and monitoring quality (Ward & Venkataraman, 1999). Standards promote understanding of the development process and communication between members. Quality models are used in conjunction with standards to clearly define attributes of a high quality software product. Also, models can also specify measures for evaluating the attributes and the entire product. In order to measure and understand quality, different models have been developed for relating quality characteristics to each other (Kitchenham & Pfleeger, 1996).

Although a detailed discussion about these models is not the main focus of this study, a brief overview goes as follows. McCall's quality model organises product quality as a hierarchy of factors, metrics, and criteria (Kitchenham & Pfleeger, 1996). McCall's model has been criticised for subjectivity in the measurement of the metrics (Côté, Suryn, & Georgiadou, 2007). Apart from the McCall's model, the International Organisation for Standardisation (ISO) also developed a software quality standard, ISO 9126, to facilitate assessment of software quality (Botella et al., 2004). It defines six quality characteristics that a high quality software product must exhibit. The attributes include correctness, integrity, usability, reliability, portability, maintainability (Ward & Venkataraman, 1999, p. 2)

2.2.3 Software Quality Management

One important aspect to the software development process is quality management. This helps in ensuring that quality standards are achieved (Gill, 2005). Software Quality Management aims at defining processes for achieving quality while Software Quality Assurance reviews and ensures that the processes are being followed (Runeson & Isacsson, 1998). A quality management system must be documented in order to achieve effectiveness (Gill, 2005). The

documentation should clearly detail standards and procedures that a development team must observe and adhere to. Quality management initiatives must focus on ensuring that the team recognises the factors that affect quality and take corrective measures. This requires upfront establishment of quality requirements and subsequent monitoring for early detection and correction of deviations. Finally, the initiatives must provide evidence that the system meets required quality standards.

Three approaches to software process quality management are: Total Quality Management (TQM), ISO 9000, and CMMI's quality management. These approaches are discussed next.

Total quality Management

TQM "represents a style of management that is aimed at achieving long-term success by linking quality with customer satisfaction" (Kan, Basili, & Shapiro, 1994). Software quality requires strong management commitment and management should take responsibility to ensure that quality is built into the processes (Ward & Venkataraman, 1999). Different techniques include the use of statistical process control that provides insight into the development process (Ward & Venkataraman, 1999). These techniques require that software quality be measured and findings reported so that improvements could be made (Ward & Venkataraman, 1999). TQM main elements are: customer focus, process, human side of quality, and measurement and analysis (Kan et al., 1994).

ISO 9000

The ISO 9000 standard is based on the premise that "a right production and management system produces the right product" (Ward & Venkataraman, 1999, p.5). The standard relies on heavy documentation for all the processes and procedures to promote control, auditability, verification / validation and process improvement. Gill (2005, p. 2) ISO 9000 standard specifies quality assurance aspects that best viewed as a network of interrelated processes. The ISO 9000/IS 14000 standard is not prescriptive on how the implementation of a system must be handled, and leaves these details for developers. This standard takes a holistic view to quality management and puts the responsibility on all people involved with the development of the software.

CMMI

The Capability Maturity Model Integration (CMMI) defines an approach to software process maturity across five levels of maturity. An organisation has to achieve process goals stated at each level before it can move to the next level (Ward & Venkataraman, 1999). As the organisation moves up these levels, it software process improve. The five levels are: Initial, Repeatable, Defined, Managed, and Optimising. As with the ISO 9000 standard, the CMMI puts emphasis on process monitoring and continuous improvement. Unlike the ISO 9000, CMMI is exclusively applicable to software development processes only (Ward & Venkataraman, 1999). CCMI has grown to be a *de facto* standard in the software processes arena (Runeson & Isacsson, 1998).

These traditional approaches to software quality have been criticised for their assumption that process improvement leads to product quality improvement (Côté et al., 2007) . This assumption leads to organisations prioritising process improvement at the expense of creating, cultivating, and using effective product quality models. According to Kitchenham and Pfleeger (1996) evidence that adherence to process standards ensures quality products is sparse. Rather, these process standards ensure only uniformity and can institutionalize creation of poor products into an organisation (Kitchenham & Pfleeger, 1996).

One other assumption in quality management is that the quality of the final product is directly influenced by the quality of development and testing processes (Kayes, 2011). The next section therefore discusses software testing as a critical component to achieving product quality. Testing is discussed separately from Traditional Quality Assurance because it is also a critical approach to Agile software development methodologies.

2.3 SOFTWARE TESTING

As the quality of software becomes increasingly important, the processes used to support and ensure software quality are gaining momentous importance as well (Winter, Rönkkö, Ahlberg, & Hotchkiss, 2008; Gill, 2005). Software testing is one such process. Testing is done to provide

confidence in the quality of the software product. The quality of testing processes and methodologies translates to level of quality of software products (Winter et al., 2008).

Testing is a key activity in SQA (Feldman, 2005) and a very important phase in a software development life cycle (Owens & Khazanchi, 2009). According to Humphreys, as (cited in Owens & Khazanchi, 2009, p. 258) "The goal of testing in Software Development Life Cycle (SDLC) is to find and document defects". It is important to note that SQA encompasses testing and other processes as discussed in previous sections. This is contrary to a common believe in some academic and practitioner works that SQA is Testing.

Effective and efficient testing and quality assurance can be realised by ensuring that the test process, techniques, and tools are built upon the people and an enabling organisational culture. This also requires the development team to understand the customer's implied, expected, and exciting requirements (Gill, 2005). The testing team must avoid taking shortcuts in testing, reducing testing time, inadequate planning, poor user involvement, poor documentations, and poor understanding of the application environment (Gill, 2005).

Software testing requires that the organisation define test processes, test cases and test plans, testing techniques, methodologies, tools, standards, and testers. The output of software is a report that shows summary of testing results. The results are reviewed for completeness and rigour. The following are the foundations of software testing (Gill, 2005):

- Test process, test cases and test plan
- Techniques, methodologies, tools and standards
- The people and the organisation

A test plan aims at catching as many defects as possible and to provide a measure of quality (Feldman, 2005). According to Feldman, a solid test plan requires agreed-upon requirements and specifications. According to Godbole as cited in Owens and Khazanchi (2009, p. 257), test cases are written to detect undiscovered defects. Writing test cases requires that requirements be reviewed for completeness and accuracy. However, as software systems become more complex, "software testing and evaluation become more difficult and its effectiveness falls

below expectations" (Gill, 2005). Testing in traditional software development processes is different than testing in Agile methodologies. It should be noted that the aim of this study is not to give a detailed comparison between traditional and Agile testing but to present a background on the field of SQA. The next chapter provides an introduction to Agile methodologies as a precursor to a discussion on Scrum and quality achievement.

2.4 AGILE METHODOLOGIES

In 2001, a group of 17 practitioners and authors published the Agile manifesto summarised in Table 2.1 adopted from Cho (2008, p. 189). The manifesto lays down fundamental principles and values that recognised software development as an empirical process (Williams & Cockburn, 2003). It views the process as one that requires teams to constantly inspect and adapt accordingly. This contradicts sharply to the traditional way of thinking which view software development process as linear and predictable through extensive upfront planning. However, over the years, it became clear that this is highly unlikely because "requirements change, technology changes, people are added and taken off the team, and so on" (Williams & Cockburn, 2003, p. 40).

Highsmith and Cockburn (2001) state that traditional process management assumed that deviations were a result of errors and therefore focused on continuously identifying and measuring errors in an attempt to eliminate deviations. On the other hand, Agile methods aim at employing a "variety of practices for constant feedback on technical decisions, customer requirements, and management constraints" (Highsmith & Cockburn, 2001, p. 122). These methods advocate for simple designs, self management, active customer involvement, flexibility, and rapid delivery of value to the customer. It is important to note that Agile methods advocate for "just enough method", contrary to the believe that their quest for flexibility renders them anti-method (Ågerfalk, Fitzgerald, & Slaughter, 2009, p. 317).

More Valuable Items	Less Valuable Items
Individuals and interactions	Processes and tools

Working Software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Table 2.1 Agile Manifesto

In light of limitations inherent in the traditional prescriptive approaches, Agile development methodologies were designed to provide new ways of planning and managing software development projects (Li, Moe, & Dybå, 2010a). Agile methodologies are characterised by: incremental development, cooperative collaboration between customers and developers, straightforward, and adaptive development processes (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). These methodologies offer better opportunities over process oriented approaches (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

Most research studies comparing Agile and traditional approaches argue in favour of Agile in terms of its ability to provide faster development cycle and improved speed to market (16 as cited in (Li et al., 2010a)). It should be noted that Agile proponents do not claim that their approach is suitable for all kinds of projects. However, the main focus is to satisfy the customer when the software is shipped rather than at project initiation. This is made possible by the fact that Agile approaches assume that change is inevitable during the life cycle of a project (Abrahamsson et al., 2002).

Agile methodologies aim at overcoming some of the shortcomings of the traditional methodologies. The iterative and incremental nature of Agile processes put emphasis on customer involvement, and frequent software releases, shorter and faster development cycles (Cho & Huff, 2011). Research reports indicate greater customer satisfaction, fewer defects, and better adaptation to changing requirements (Cho & Huff, 2011). Highsmith and Cockburn (as cited in Abrahamsson et al., 2002, p.14) note that Agile methods recognise the value of people, effectiveness, and manoeuvrability as crucial for successful delivery of projects.

2.5 SCRUM METHODOLOGY

2.5.1 Scrum

According to Schwaber and Beedle, (as cited in Abrahamsson et al., 2002, p. 29), the first mention of Scrum in the literature referred to "an adaptive, quick, self-organising product development process originating from Japan". The following is a definition of Scrum:

"*Scrum is a simple framework used to organise teams and get work done more productively and with higher quality. It is a lean approach to software development that allows teams to choose the amount of work to be done and decide how best to do it.*"

(Sutherland & Schwaber, 2007, p. 11)

Scrum as an Agile approach emphasises flexibility, adaptability, and productivity. The focus is on the effectiveness and agility during the life cycle (Schwaber, 1995, p. 10). It allows developers to be adaptive within a complex, chaotic environment using imprecise processes. This necessitates maximum flexibility and greater tolerance for changes in environmental variables. The variables that influence release plans include customer requirements, time pressure, competition and the required level of quality given the other variables (Schwaber, 1995). Scrum is said to be an empirical process approach as opposed to a defined process approach because it makes the assumption that "analysis, design, and development processes in the Sprint phase are unpredictable" (Schwaber, 1995, p. 10).

"Scrum defines the systems development process as a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems" (Schwaber, 1995, p.1). Scrum views the software development process as a complex, unpredictable journey characterised not only by changes in requirements, but also by changes in environmental and technical variables. One of the strengths of Scrum is its focus on clearing any impediments during the life cycle (Schwaber, 1995).

The Scrum methodology provides a project management method that is suitable for situations where it is not easy to make stable plans. According to Li et al. (2010a), scrum provides greater

decision making power to developers at the operational level. It empowers teams to cope with inherent uncertainty, requirements changes, incomplete requirements specification, and ambiguity in software development (Sutherland, Johnson, & Jakobsen, 2008).

The Scrum framework is defined by three constituent parts: roles, ceremonies, and artefacts (Cho, 2008; Marchenko & Abrahamsson, 2008). These are summarised in Table 2.2.

Roles	Ceremonies	Artefacts
Scrum Master	Daily Scrum	Sprint Backlog
The Team	Sprint Planning	Product Backlog
Product Owner	Sprint Review	Burndown Chart
	Sprint Retrospective	

Table 2.2 the Scrum framework

2.5.2 Scrum Workflow

Scrum process follows an iterative and incremental approach to development (Schwaber, 2009). The process starts with a team reviewing the work it commits to do. After the commitment, the team assumes independence to devise the best ways of achieving the increment. At the end of the iteration, the team presents the work to stakeholders for inspection, review, and adaptations. The Scrum process involves three phases: pre-game, development, and post-game as shown.

The pre-game phase involves planning and architecture design sub-phases (Abrahamsson et al., 2002).The planning sub-phase defines the system being built. This involves creating a Product Backlog that contains all requirements that are known at the time. The Product Backlog items are amenable to changes and additions as more or new requirements come to the fore. It also includes prioritisation and estimation of the resources needed to implement the requirements. The architecture design sub-phase produces a high level design of the system and the architecture based on the Product Backlog.

The development phase "is the Agile part of the Scrum approach" (Abrahamsson et al., 2002, p. 31). It involves constantly observing and controlling the various technical and environmental variables that might change. The development of the actual system happens in Sprints with each Sprint representing a two to four week iteration that produces a new working increment to the system. Details of each Sprint are decided during a two-phase Sprint Planning meeting whereby items to be developed during the Sprint are listed in a Sprint Backlog (Abrahamsson et al., 2002). The Sprint Planning Meeting "begins with the Product Owner reviewing the vision, the roadmap, the release plan, and the Product Backlog with the Scrum Team" (Sutherland & Schwaber, 2007, p. 13).

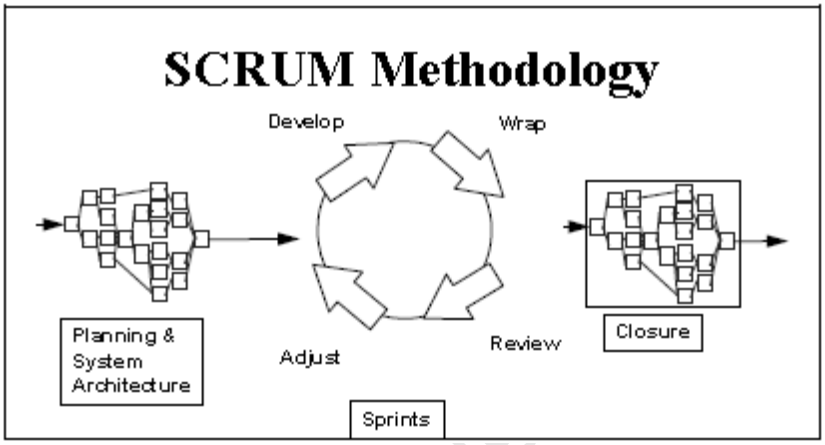


Figure 2-1 Scrum methodology (Schwaber, 1995, p. 10)

Each increment represents properly tested and well written code converted into an executable. The user operation of the functionality in help files or user manuals should accompany the executable code. "This is the definition of a done increment and it should factor into how much work a team can take in a Sprint" (Schwaber, 2009, p.2)

There are important characteristics to note about the Scrum methodology shown in Figure 3. The first characteristic is that, the Planning and Closure phases are linear and well defined, both in terms of processes, inputs and outputs (Schwaber, 1995). The Sprint phrase is empirical in nature, undefined and uncontrolled, and requires external mechanisms to control and mitigate risk, and improve flexibility (Schwaber, 1995). According to Koch (2005), the short increments

that characterise Agile methods act as a risk mitigation mechanism. The Sprints are not linear, and sometimes require trial and error to build knowledge in the process of evolving the product (Schwaber, 1995). As mentioned before, the project remains open to environmental influences such as time pressures and quality requirements during the Planning and Sprint phases.

2.5.3 Scrum Roles

Scrum defines only three roles namely: Product owner, ScrumMaster, and Team (Sutherland & Schwaber, 2007; Marchenko & Abrahamsson, 2008; Schwaber, 2009; Schwaber, 2004). A brief summary is given in Table 2.3.

Role	Brief Description
Product Owner	responsible for defining, prioritising, and changing product features, and specifying release dates among other responsibilities
ScrumMaster	The ScrumMaster is a team leader and facilitates productivity, collaboration between role players, resolving impediments, and coordinates the whole development process. The ScrumMaster needs to have proper training to run efficient Scrum meetings and provide relevant coaching to the team
Team	The team ideally has between five to nine members. This self-organising and cross-functional team is responsible for development and delivery of project goals. The mandate is to achieve the goals set for each Sprint by autonomously setting its own pace and development approach

Table 2.3 Scrum roles

2.5.4 Scrum Ceremonies

Scrum prescribes four different ceremonies that should be adhered. These ceremonies occur at the beginning of a Sprint, during the Sprint, and at the end of the Sprint. Table 2.4 gives a brief summary of these cermonies (Abrahamsson et al., 2002; Sutherland & Schwaber, 2007; Marchenko & Abrahamsson, 2008).

Ceremony	Description
Sprint Planning	The Sprint Planning meeting also involves gaining commitment from developers that they will achieve the goals of the Sprint.
Daily Scrum	Daily Scrum meetings are held during the Sprint to discuss progress and address impediments.
Sprint Review	This meeting gives participants an opportunity to assess the increment and redefine the direction of project. When the development of the system is complete and agreement between involved parties is reached, the project enters the post-game phase.
Retrospective	Retrospective is an opportunity for a team to identify ways in which the development process can be adjusted and thus improved. The main aim of retrospective should be continuous improvement which is an ideal that must be commonly shared by all team members

Table 2.4 Scrum Ceremonies

2.5.5 Scrum Artefacts

Scrum introduced a few artefacts used throughout a development process as described in Table 6.3. (Marchenko & Abrahamsson, 2008; Schwaber, 2009).

Artefact	Description
Product Backlog	The product backlog contains technical and business functionality to be developed. The Product Owner controls the list and is in charge of properly prioritising the list. The product backlog is never complete.
Burndown Chart	Depicts the amount of work remaining across time and progress made.
Sprint	Contains the work to be done in a Sprint. Unlike the product backlog, the Sprint

Backlog	backlog cannot be changed for the duration of the Sprint.

Table 2.5 Scrum Artefacts

Scrum works differently from traditional process control mechanisms which rely on detailed plans, Gantt charts, and work schedule (Schwaber, 2004, p. xvii). The next section reviews the literature on Scrum and software quality.

The next provides an overview of disparate empirical field studies on various aspects of Scrum.

2.6 DISPARATE FIELD STUDIES ON SCRUM

2.6.1 Overview

Although there is an increasing interest in Agile research from the academic community, the field is still dominated by practices emerging in industry. According to Dingsøyr et al. (2012) research on Agile methods has primarily focused on issues around adoption, test driven development, pair programming, team dynamics, and challenges in distributed environments. More research is therefore needed to shed light into how these appealing practices could produce desired outcomes as claimed by practitioners (Abrahamsson et al., 2009).

2.6.2 Studies on Enablers of Quality

A few studies have investigated the use of quality achievement techniques in Scrum. Li et al. (2010) found that constant collaboration amongst developers, early testing, iterative development and continuous feedback improve defect correction. These findings were also made by Sutherland et al, (as cited in Li et al., 2010). In addition to early testing, the authors also concluded that knowledge sharing, retrospective, and daily meetings helped to improve defect fixing efficiency. Another study, Mnkandla and Dwolatzky (2006), concluded that allowing no changes until the end of a Sprint in Scrum is a way of preventing scope creep.

(Green, 2012) reports a product quality improvement after adoption of Scrum as a result of value-driven decomposition of features, a clear Definition of Done, testing expertise, adherence to the Definition of Done. Green acknowledges the role of engineering practices from XP and top management direction in allowing in achieving better quality products.

In a rating of teams at Adobe Systems, (Green, 2011) discovered that teams that adopted Scrum without changing their processes to accommodate key practices, roles, and artefacts did not achieve fruitful effectiveness. According to (Green, 2011) , proper adoption of Scrum lead to improvements in quality and this improvement was made possible by assigning quality assurance responsibilities away from SQA teams to development teams. Successful implementation was also found to have a strong correlation with improved defect management.

Cho (2008) found that a majority of developers valued the improved communication between team member in Scrum and the increased customer involvement. In addition, the Scrum ceremonies were identified as being very helpful. Caballero, Calvo-manzano, and Feliu (2011) studied how an organisation attempted to improve project productivity by introducing Scrum to assess the impact on efficiency and product quality standards. Their finding is that that Scrum can improve productivity without compromising quality in a very small enterprise.

2.6.3 Studies on Challenges to Achieving Quality

While Scrum promises to offer greater control over development processes and improve quality, Sutherland et al.(2008) found challenges faced in Scrum implementations. The implementations in the cases studied do not meet basic Scrum requirements. For example, the process does not result in fixed Sprint iterations to produce fully tested and working software. In some cases, the product owner role is not clearly defined. In other cases, teams do not keep a Burndown chart and do not know the velocity of software production thus making it hard for the Product Owner to make release plans (Sutherland et al., 2008). As such, Sutherland et al. (2008) advise software development organizations to consider introducing CMMI practices to improve process maturity while using Scrum. In particular CCMI Level 3 list generic practices that can be helpful to Agile software development.

According to Rong, Saho, and Zhang (as cited in (Caballero et al., 2011) the productivity and quality in Scrum depends on the talent and capabilities of team members. Another weakness noted by Timperi (2004) Scrum leaves too many aspects about quality management open and

it needs practices from other Agile methodologies. If scrum is used independently it is suitable in cases where validation is emphasised and verification not important (Timperi, 2004).

Cho (2008) found that the reduced documentation in Scrum and other Agile methodologies made it difficult for some developers to work with existing code for the first time. They also found that due to lack of documentation, new developers ask a lot of questions thus wasting valuable time for the senior developers. Further, Cho also discovered that some developers had a problem with collocation and working in an open plan. The problem is with distraction that makes it difficult for developers to concentrate.

Scrum has been found to exert excessive time pressure and stress on developers thus making it hard for them to carry out certain tasks such as refactoring (Li, Moe, & Dybå, 2010b). In this case study, the authors also found that iterative development and early testing in Scrum helps improve defect management. This study is one of the very few that focus on Scrum and aspects of software quality assurance. The focus in this study is more on defect management and defect density by comparing defect data from pre-Scrum phase with defect data from post-Scrum phase.

Akif and Majeed (2012) identified a long list of challenges and issues encountered in Scrum implementations. However, a majority of the issues identified and attributed to adoption of Scrum seem to have been a result of poor implementation and less informed conception of Scrum.

In summary, research on software quality faces the challenge of providing tools and technologies that will help industry to use safe, dependable, and usable products within an economic framework (Osterweil et al., 1996). One major challenge facing Agile researchers is to ensure that their studies embrace both research rigour and relevance to industry (Dingsøyr et al., 2008). Research rigour implies that the output of the study provides a better understanding of the field. In order to achieve this, theories from other related fields should be employed. Dingsøyr et al., (2008) advise that using methodologies such as grounded theory can help achieve middle range theories.

This study aimed to make contribution to both industry and academia by producing a middle range theory that is grounded in data. The anticipated output of the research was greater understanding of Scrum methodology from a quality assurance perspective. The study involved collaboration with practitioners in the field.

The next chapter presents the research methodology which guided the study.

CHAPTER 3. RESEARCH METHODOLOGY

3.1 METHODOLOGY

This chapter presents and justifies methodological choices employed to build an exploratory substantive theory on aspects of Software Quality Assurance (SQA) in Scrum. According to Strauss and Corbin, (as cited in Tan, 2010, p. 99) a research methodology is a "way of thinking about and studying social reality and a research method is a set of procedures and techniques for gathering and analysing data". A methodology therefore entails ontological and epistemological assumptions about social reality. This chapter presents the research paradigm, the research method, the research strategy, and the nature of data used. This chapter focuses on the debates surrounding the methodological assumptions employed in this study, and the reasoning behind the choices. The actual detailed specifics of how these 'choices' were used to guide this study are presented in the Research Approach Chapter.

3.2 THE GROUNDED THEORY METHOD

According to Goulding (as cited in Matavire and Brown, 2011, p. 2) "a grounded theory is defined as theory which has been systematically obtained through social research and is grounded in the data". Lehmann (2010) also backs this definition. While GTM is different from general Qualitative Data Analysis (QDA) many of its tenets are not unique to it. Lehmann (2010), states that the most prominent difference between GTM and other qualitative methods is GTM's approach to rigour. The difference between GTM and other qualitative methods can be explained by the following tenets of GTM:

- "Theory is emergent from empirical data rather than from inferences or existing theories
- The constant comparison method enable theory generation during systematic collective and analytic procedures
- Memo writing is the formulation and revision of theory throughout the research process
- The research process is flexible and creative" (Tan, 2010, p. 95).

Lehmann (2010) states that GTM places more emphasis on iterative data collection and analysis and these iterations are much tighter than in other qualitative methodologies. The tight coupling between data collection and analysis underpins theoretical sampling which proceeds until no new insights emerge from the data. For data analysis, it is crucial that each incident of data is compared instantly with all other incidents collected so far and this procedure must be followed religiously (Lehmann, 2010).

The next section discusses the two strands of GTM by its founders and presents the preferred rendition used by the researcher.

3.2.1 Glaserian versus Straussian GTM

Since the original publication of GTM, the method has been marred by disagreements between its founders (Roode & Niekerk, 2009; Matavire & Brown, 2011). The debate surrounds the canons of the methodology and the creators of the methodology disagreed on the right tenets of the methodology. This debate has resulted in two alternative approaches namely 'Glaserian' and 'Straussian' approaches (Matavire & Brown, 2011). Glaser emphasizes adherence to the original way of GTM and prefers the term Classic GTM. On the other hand, Strauss teamed up with Corbin to produce an evolved form of GTM referred to as Evolved GTM. The differences between these two approaches are articulated in (Roode & Niekerk, 2009) and the researcher does not intend to engage in the debate between the methods as this has been extensively dealt with in the literature.

For the objectives of this study, the researcher opted to follow the Classic GTM as the best approach to generate an exploratory and grounded account of aspects of SQA in Scrum for the reasons provided in the next section.

3.2.2 Classic GTM

Glaser emphasized theory emergence by data conceptualization, whereas Strauss and Corbin "introduced a new coding process with a strong emphasis on conditions, context, interaction strategies and consequences" (Tan, 2010, p. 95). Glaser objected strongly to this and believed it was 'forcing' of the concepts as opposed to allowing the theory to emerge (Urquhart et al., 2010). The lack of prescriptions in Classic GTM is considered to be a strength because it offers researchers flexibility and creativity for conceptualisation (Roode & Niekerk, 2009). This is one of the main reasons why the researcher opted to use Classic GTM over other approaches. The researcher believed that going into a previously unexplored territory requires maximum creativity and freedom. The researcher believed that creativity and systematic application of a unified whole of Classic GTM tenets would result in a rigorous theory generating process than a forced confinement.

The Classic GTM researcher has a range of theoretical coding families to use for conceptualising how categories relate to each other (Hoda et al., 2012). The coding families include: The Six C's (causes, contexts, contingencies, co-variances, and conditions), Process (stages, phases, passages etc), Degree family (limit, range, intensity, etc), Dimension family (dimensions, elements, divisions) and many more. Glaser did not prescribe any specific coding paradigm to be followed, and espouses that a researcher can add to the coding families (Roode & Niekerk, 2009). It is important to note that according to Glaser (as cited in Hoda, Noble, & Marshall, 2012, p. 624) theoretical codes are not strictly required although they enhance the quality of a GTM product. It was therefore important for the researcher not to be restricted or confined to any particular paradigm because the research objectives were kept as broad as possible from the on-set and the researcher wanted to remain open to the emerging theory.

Furthermore, the fact that the study was conducted in a relatively unexplored territory means the researcher needed to be as open to emergence as possible. This is more achievable through the use of Classic GTM over other approaches because with Classic GTM, the researcher does not start with a research question but investigates a main concern of subjects (Roode & Niekerk, 2009). This allowed for emergence of concepts that had previously not been uncovered in Scrum and SQA. This also allowed the researcher to remain open to what the data was indicating and, through constant comparisons, the researcher arrived at grounded concepts that illuminate aspects of SQA in a Scrum environment.

The Classic GTM is not without criticism though. Roode and Niekerk (2009) state that the Classic GTM does not follow traditional research which makes it difficult for researchers to justify the need for their studies. In addition, authors such as Suddaby (as cited in Matavire & Brown, 2011) state that it is illogical to conduct reasonable research without a clearly defined research question. Further, the Classic GTM mandates that researchers must not start with research questions and extensive literature review (Roode & Niekerk, 2009) in order to avoid preconceptions that may lead to theory testing instead of theory building and this goes against conventional university post-graduate research requirements. Glaser defends Classic GTM against the criticism by advising researchers to use all the tenets as one package instead of picking some and omitting others (Glaser & Holton, 2004).

It is important to note that the recommendation by Glaser does not mean that researchers should completely discard literature review before taking up the research work as most studies are supposed to be situated in the context of current research work (Adolph, Kruchten, & Hall, 2012). For this study, the researcher also had to provide a descriptive account of Scrum and SQA as a basis for further discussion and as a sharpening mechanism for effective conversation with practitioner participants (Hoda, Noble, & Marshall, 2012). Furthermore, the researcher needed to conduct the literature review in order to stimulate ideas for research and identify gaps in the extant literature on Scrum and SQA. The reviewed literature helped strengthen the realisation that the IS research community has given little attention to the Scrum methodology. In addition, the researcher had to provide literature review as part of requirements by the University of Cape Town for partial fulfilment of a Master of Commerce degree.

3.3 RESEARCH PARADIGM

Ontology is concerned with "whether social and physical worlds are objective and exist independently of humans, or subjective and exist only through human action" (Orlikowski & Baroudi, 1991, p. 8). Ontological assumptions inform epistemological beliefs. An epistemology represents as a set of philosophical assumptions about the nature of phenomena and how valid knowledge about these phenomena may be generated (Orlikowski & Baroudi, 1991). Epistemology is concerned with "what is knowledge" and "how we obtain valid knowledge" (Hirschheim & Klein, 1985). The three main epistemological perspectives in IS research are: Intepretivism, Positivism, and Critical Research. These philosophical perspectives are discussed in detail in (Orlikowski & Baroudi, 1991).

While IS research had traditionally followed the positivist paradigm (Orlikowski & Baroudi, 1991), the researcher found it appropriate to follow the interpretive paradigm in order to achieve the research objectives. In addition to the research objectives, the interpretive paradigm is compatible with GTM because GTM can be used in any paradigm (Urquhart, 2001; Lehmann, 2010). The interpretive paradigm was particularly attractive because it makes it possible "to understand how members of a social group, through their participation in social processes, enact their particular realities and endow them with meaning, and to show how these meanings, beliefs and intentions of the members help to constitute their social action" (Orlikowski & Baroudi, 1991, p. 13). This was important for this study as the aim was to understand aspects of SQA work in Scrum from the perspectives of the practitioners at SAIT.

3.4 THE CASE STUDY STRATEGY

"The case study is a research strategy which focuses on understanding the dynamics present within single case settings" (Eisenhardt, 1989). Klein and Myers (1999) state that case study research is now a valid and accepted strategy in IS research. It is important to note that the researcher sought to gain deep understanding on the phenomenon and therefore opted for a single-case design.

The combination of GTM and case study methodology offered several advantages. As stated previously, both these methodologies are suitable for exploring emerging areas. Further, the use of comparative analysis of evidence allows researchers to reconcile conflicting or paradoxical evidence (Eisenhardt, 1989; Fernández, 2004). This proved critical when the researcher had to confront differences in the meanings and value attached to different SQA practices by the participants. Comparative analysis also helped to implicitly build validity into the findings because the data was questioned from the beginning to the end. In addition, the close connection between the data and theory renders the theory amenable to further testing and expansion or falsification (Eisenhardt, 1989). Finally, one adavantage of the case study methodology is 'face validity', of which, according to (Myers, 2009), implies a representation of a real life story. According to Myers, plausibility is the key factor in interpretive case studies and this is contrary to the focus on validity and reliability in positivist studies.

3.5 RELIABILITY AND VALIDITY IN CASE RESEARCH

Fernández (2004) advises researchers using both GTM and case study methodology to explicitly state the methodology driving their research work. It is important at this juncture to state that this research study followed GTM canons not canons of the case study methodology. According to Fernández (2004) it is imperative that true emergence of theory is not distorted by tenets of the case study methodology.

For achieving reliability and validity, Glaser (2004) recommends the use of GTM's procedures as a methodological whole whereas Yin, as cited in (Voss, Tsikriktsis, & Frohlich, 2002) espouses the use of pattern matching, explanation building, time-series analysis, use replication logic, case study protocol, and case study database. According to Glaser as cited in (Adolph, Hall, & Kruchten, 2011, p. 495) validity in GTM is achieved after "much fitting of words, when the chosen one best represents the pattern". Klein and Myers (1999) argue that the criteria for evaluating case study research as stipulated in (Benbasat et al., 1987) and (Voss et al., 2002) are inappropriate for interpretive research as followed in this study. Last but not least, interpretive case studies have been mistakenly criticised for lack generalizability (Klein & Myers, 1999). This study closely adhered to GTM's canons as demonstrated in the research approach chapter.

It is therefore important that GTM studies are not subjected to validity and reliability concerns as mentioned in Voss et al., (2002).

Apart from case methodology demands, Glaser (2004) states that a GTM product is not a "factual description" which is often a demand in pure Qualitative Data Analysis (QDA). Glaser emphasises that GTM is not concerned with producing accurate and descriptive facts demanded in QDA. The main focus is on producing plausible and grounded conceptualisation integrated into theory. The theory is open to falsification, modification, and/or extension through further constant comparative analysis.

GTM canons such as comparative analysis and simultaneous data collection and analysis act as control mechanism to curb researcher bias (Fernández, 2004). Theorectical Meoming is another tenet of GTM (explained in Research Approach Chapter) which curbed deep-seated assumptions from the researcher about the substantive area. These tenets and how they have been employed in this study are clearly articulated in the Research Approach Chapter on data collection and analysis. It is also important to note that the GTM tenets can only serve to limit researcher bias but cannot completely prevent it.

3.6 QUALITATIVE VERSUS QUANTITATIVE DATA

According to Glaser (2004, p. 9) GTM can use any data because it is independent of the nature of data (Lehmann, 2010) and should not be restricted to any ontologies/epistemologies (Lehmann, 2007). For the purpose of this study the researcher decided to use qualitative data through interviews in order to gain richness from textual accounts. Appendix A provides a base guideline used in all interviews.

According to Myers (1997), qualitative research requires collection of qualitative data through mechanisms such as interviews, documents, and participant observation. Interviews formed the major mode of data collection for this study. There are three types of interviews that available to researchers (Myers, 2009). These are: structured interviews, semi-structured interviews, and unstructured interviews. According to Myers (2009), unstructured interviews employ very few pre-formulated interview questions. Structured interviews require the use of, and adherence to

pre-formulated interviews. Semi-structured interviews offer an opportunity to pre-formulate interview questions as well as some degree of openness during the interview. The researcher used semi-structured interviews in most cases. The Research Approach Chapter provides more details on the actual data collection specifics during the course of this study.

The researcher drew inspiration from Glaser (2004) in his quest to dispel strong emphasis on data accuracy and trustworthiness as being concerns of the mainstream qualitative data analysis. For this study, field notes were used only as a commentary after team meetings or ad hoc conversations. It is important to state that it was also not easy to capture impressions through observations on quality assurance processes because the researcher was not a member of the Portal Project. However, the researcher had many discussions over lunch time with friend developers about various aspects of SQA emerging from the data such as absence of a dedicated tester, code reviewing, and developer testing. In some cases, most of what they said was already captured in prior formal interviews and did not add anything analytically to the study therefore their points were not written down.

CHAPTER 4. CONTEXT DESCRIPTION

4.1 CASE ORGANISATION

The research took place at SAIT, a South African organisation in the Information Technology (IT) services sector. SAIT specializes in business and IT consulting. The organisation began operations in 1997, has branches in the United Kingdom, and is head-quartered in South Africa. SAIT's key offering is to partner with clients by providing expert IT solutions to business

problems and IT-enabled change programmes which often involve business analysis and software development. In 2011, the organisation had 150 full time employees in its payroll.

In the software development services arena, SAIT provides technology consulting which includes software product development and enterprise architecture consulting to big and medium size organisations. It also develops software for its internal business areas. The department responsible for delivery of software development services is known as Software Services Delivery (SSD). At the time of data collection, SSD had four software development teams working on different client and internal projects using Microsoft Dot.Net and Java technology platforms.

4.2 PROJECT BACKGROUND

In 2006, departments at SAIT expressed the need for development of a custom solution to consolidate internal business processes as well as streamline client-specific contractual processes. This lead to the development of a small employee management system for general human resources functions and was called Staff Information Database (SID). In addition to serving specific business needs, SID was also used as a proof of concept application for the then new Google Web Toolkit (GWT) framework for developing rich internet applications. It was also used as a 'playing-ground' for Java developers who were not contracted to a specific client project. At the time of this study, the Portal Project was still under development.

In 2010, the organisation decided to revamp the SID and strategise its development in a modular approach which would lead to design and development of an organisational portal – the 'Portal Project'. From a technical perspective, SID's rich functionality had outgrown its initial design and needed architectural redesign.

The aim of Portal Project is to allow for the efficient sharing of data and allow for easy interactive functional behaviour to be readily available. The idea is to give employees an opportunity to make a meaningful contribution to their workplace. The 'Portal Project' development project is the main focus of this study. The development team for this project will be referred to as *the project team*.

As part of a broad strategy, the management at SAIT decided to commoditize the product by customizing and installing cloud based instances for different clients on demand. The way it works is that each client selects modules of the system that it wants to use and specifies custom functionalities or features that it wishes to have. The instance is customized to meet the client's requirements. In most cases, the client instances comprise of changes to themes, colours, and screen layouts. The core functionality of selected features is almost always the same with minor tweaks to suit client specific business processes.

4.3 THE PROJECT TEAM

A Project Board was set up to drive the vision of the product when the organisation decided to strategise Portal Project in 2010. This vision has been driving the product backlog. The CEO heads the Project Board and is the official Product Owner. The core development project team is made up of Computer Science and Information Systems Honours graduates from leading South African universities. The project team roles were as follows: eight programmers, one software analyst, one business analyst who 'shadows' the Product Owner, two user experience (UX) developers, and the ScrumMaster. It is important to note that there was no dedicated tester at SAIT. The team is responsible for all testing. Table 4.1 gives a breakdown of the project team.

In addition to their academic qualifications, the developers had each written a number of Java certification examinations such as Sun Certified Programmer for the Java 6 Platform, Sun Certified Web Components Developer for the Java 5 Platform, and Core Spring Training. This team constantly engaged in identifying and researching new technology frameworks to improve its toolset. For example, towards the end of the study, one lead developer had just completed research on testing strategy and how to incorporate and follow Test Driven Development.

Job Position	Project Team Role	Experience
Senior Software Project Manager	ScrumMaster	11 Years
Application Architect	Developer	12 Years
Lead Developer 2	Developer	7 Years

Lead Developer 2	Developer	6 Years
Developer 2	Developer	4 Years
Developer 2	Developer	4 Years
Developer 2	Developer	2 years
Developer 1	Developer	1 Year
Developer 1	Developer	1 Year
Software Analyst	Software Analyst	3 Years
Business Analyst	Product Owner	4 Years
Lead Developer 2	UX Developer	6 Years
Front End Developer	UX Developer	3 Years

Table 4.1 Team Composition

4.4 THE HISTORY OF SCRUM AT SAIT

SAIT piloted Scrum in 2010 as part of their broader change in strategy. Portal Project was to be used as a pilot project to introduce Scrum in the organisation. The release for Sprint 1 was scheduled for July 2010. The core team started scrumming officially on the 21st July 2010. Prior to that, the organisation employed a waterfall-based development methodology. The rationale behind the move to Scrum was to curb some of the difficulties experienced with waterfall model on SID and client projects. Some of the frustrations on waterfall based development in the context of SAIT can be seen from the Wiki extract below written by one of the key developers in 2010:

"Current ERM Portal development for internal projects is waterfall based:

1. *Consultants have workshops with the client and write up a spec*
2. *Developers implement the system using very little from the spec (lots is missing, what is there has not been thought through well enough, client doesn't know what they want/need yet etc.)*
3. *Developers leave the project and go onto higher priority billable work*

4. *Consultants and the client go through a formal test cycle and want changes and bug fixes but very little development time is available"*

The organisation therefore decided to test Scrum on its internal project before using it on client projects.

4.5 SCRUM IMPLEMENTATION BY THE PORTAL PROJECT TEAM

The project team GreenHopper software as a tool for visualizing Scrum project management. The software provides a Planning Board functionality that allows teams to manage backlogs for each iteration. It offers dragging functionality that allows users to order items, and move them across stages of development. It allows members to break down stories into technical tasks reflecting estimate, status, and version. The tool offers the ability to track and monitor burndowns and schedules. In addition to GreenHopper, the team also uses Git for version control and hosts the project on GitHub. The team also uses Hudson for continuous integration.

The Scrum process workflow consists of five phases: New, Ready, In Progress, Ready to be Tested, In Testing, and Complete. This workflow reflects steps through which a Story has move before it is declared as 'Done'. The 'Ready to be Tested' and 'In Testing' steps are particularly important steps to quality development. It is in these two steps where code reviews and developer testing take place. A story has to be code reviewed before it can be developer tested. After code review feedback has be incorporated, the story is tested, and marked as 'Complete' if the testing did not uncover bugs.

Each Sprint takes 2 weeks, at the end of which follows a retrospective meeting to determine what worked well, lessons learned, problems, and areas of improvement. At the beginning of each Sprint, which is normally on a Wednesday, the team holds Sprint planning and review meetings. During this meeting, work done during the Sprint is analysed and progress is reflected by the Sprint Burndown chart. The Project Board and any interested people attend the review meeting.

The Sprint planning meeting is divided into two parts. Part I focuses on assigning work to the Sprint, sizing backlog items, and determining resource capacity. Part II focuses on identifying

tasks that the team commits to undertaking during the Sprint and producing the Sprint Backlog. Before the planning meeting, the Scrum Master and Product Owner should have produced the Sprint Backlog.

CHAPTER 5. RESEARCH APPROACH

This chapter presents a confessional account (Schultze, 2000) of the research process. It details how the research process unfolded from the beginning to the end. The main focus is on the application of an integrated set of Classic GTM tenets throughout the research process. The theorisation process was not straight forward and was non-linear. Figure 5-1 depicts how the research process explained in this chapter unfolded. This figure is rendition of a figure in Hoda et al. (2012) adopted and modified here to provide more detail and clarity on how this particular research process enfolded.

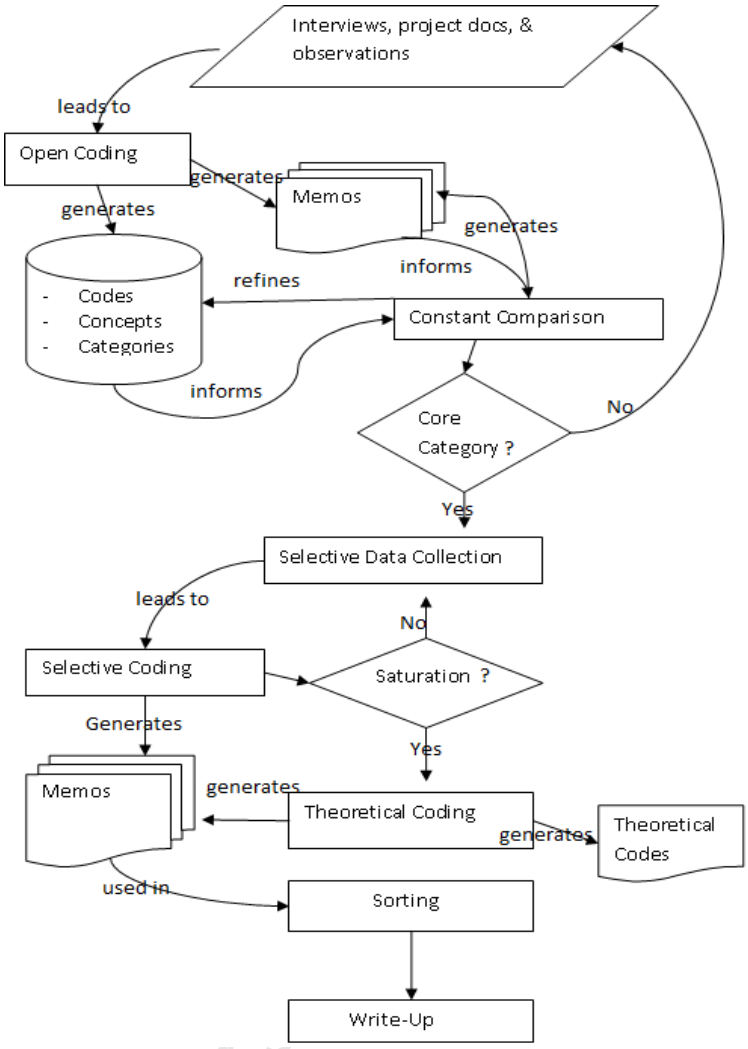


Figure 5-1 The Research Process: Applying Classic GTM – adopted from (Hoda et al., 2012)

Section 5.1 details data collection while Section 5.2 details general Classic GTM tenets such as Constant Comparison. Section 5.3 details the Data Analysis process through the three levels of coding while Section 5.4 provides an overview of theoretical saturation, delimiting of the theory, and theoretical sorting.

5.1 DATA COLLECTION

In retrospect, data collection comprised three phases: an initial exploratory phase (Phase 1), the formal phase (Phase 2), and the selective phase (Phase 3). It is important to note that these phases were not designed in advance but were guided by the emerging theory. Table 5.1 provides a list of participants, their respective roles, and the data collection phase in which they

participated. It is important to note that the researcher did not focus solely on members of the Portal Project. Some participants, such as Participant 9 and Participant 8 were not members of the Portal Project.

The researcher started discussing his intentions with senior stakeholders (software development leadership) in October 2011. Phase 1 took place from October 2011 to early 2012 through informal meetings. This exploratory phase started with conversations with two initial participants where the researcher discussed areas of concern within SAIT. The participants emphasised software testing as an area of concern. The researcher decided to explore more options and organised an 'initiation' meeting with three senior employees: the Head of Software Operations, the ScrumMaster, and the Support Manager. The purpose of that meeting was to discuss the broad area of interest and to gain insights into what particular areas were pressing to SAIT. Options emerging from the meeting included focusing exclusively on the role of testers in Scrum, the impact of dedicated testing on productivity, customers' perceived quality of products from SAIT, and Software Quality Assurance.

After the first initiation meeting, the researcher explored documents that detailed the context of the Portal Project, the history, and the implementation of Scrum. These documents provided an understanding of the case organisation (SAIT) and of the Portal Project. Lessons learned from this phase helped narrow down the research problem. This is valid in Classic GTM because "the researcher works with a general area of interest rather than with a specific problem until a problem is identified" (Adolph et al., 2012, p. 1271). The outcomes of Phase 1 included clearly articulated research objectives, an understanding of different SQA issues important to SAIT, and list of possible participants for interviewing.

Formal interviewing, transcribing, and data analysis (Phase 2) and Phase 3 took place from March 2012 until end October 2012. These two phases comprised of formal recorded interviews and ad hoc unrecorded conversations. Phase 2 comprised of eight formal and recorded interviews. The number and the choice of participants for these phases was not pre-conceived. The researcher was guided by theoretical sampling (explained in Section 5.1) and the emerging theory. According to Adolph et al. (2012, p. 1273) "the sampling strategy must

promote the development of sufficient concepts to support a conceptually dense grounded theory". Theoretical sampling provided an opportunity to assess the emerging concepts and decide on the next participant. Deciding on the very first participant for the formal interviewing phase was easy because it was based on the information gathered during Phase 1. This decision can also be seen as a judgemental sampling (Marshall, 1996) and also as opportunistic because Participant 1 had been part of Phase 1, was about to leave SAIT, had interest in software testing, and the researcher had grown 'comfortable' around him.

Throughout Phase 1 and Phase 2, the Head of Software Operations acted as a co-researcher and had vested interest in the thesis. He was also available to discuss different ideas that he had on improving the software quality approach at SAIT. He was always willing to communicate his vision and the problems he had noted since joining SAIT and became the main source of data during the first phase. He also instructed the team leaders to give the researcher access to both the GreenHopper software (visualisation of the Scrum process) and Git-Hub mailing list in order to have access to all mails shared after code reviews.

The interview questions were always tailored for individual participants based on their specific role in the team, the level of seniority and their experience. This helped because senior participants were more aware of issues around context of development, the rationales behind the development process and SQA practices, and the reasons behind presence or absence of some important SQA resources such as a Dedicated Tester. The junior participants were helpful in giving insights about operational matters, learning aspects, and experiences during various Scrum ceremonies. This approach helped maximise the richness of data from each interview.

Tailoring questions for individual participants does not mean the questions were overly different. The theme in the questioning was always the same as stipulated in Appendix A. Each interview took its own trajectory depending on the openness of interviewee and atmosphere. Some participants were calm and relaxed and open to discuss. Others were more defensive and it was difficult to coax information from. After the third interview, the researcher decided to ask participants for a 'chat about my thesis', not an interview. The researcher also stopped

carrying a printed list of questions to the interviews. This improved the richness of data collected because most participants were warm and welcoming.

Prior to each interview, the researcher detailed specific information to be collected based on the needs of the emerging theory, insights from prior interviews, and areas marked for further exploration in prior interviews. After each interview, the researcher captured memos about impressions of the interview. As the data collection evolved, the researcher used ideas and/or questions captured in memos and the emergent story to decide on next data to collect. The memos captured questions about gaps on the emerging theory, areas that needed further clarification and areas that had seemingly conflicting evidence. For example, Participant 6 had voiced concerns around the nature of code review process in the organisation. The participant was not satisfied with the fact that some developers liked to impose their specific style of writing code in the feedback they gave during code reviews. The researcher followed this up in subsequent interviews and learned of '*the diminishing value of code reviews*' which is one of the properties of the ***Code Reviews*** concept of the ***Adopted Practices*** category (Details on the categories can be found in the Findings Chapter).

Phase	Participant	Job Position	Experience (Years)
1	(Exploratory)	Head Of Software Operations	21
1	(Exploratory)	Business Analyst	2
1 & 2	Participant 1	Business Analyst	5
2	Participant 2	Senior Business Analyst	7
2	Participant 3	App. Architecture	10

2	Participant 4	Senior Software Project Manager	9
1& 2	Participant 5	ScrumMaster	11
2	Participant 6	Lead Developer 2	7
2	Participant 7	Business Analyst	2
1 & 2	Participant 8	Support Manager	7
3	Participant 9	Head of Regional Operations	11
3	Participant 10	Business Analyst	4
3	Participant 11	Lead Developer 2	6
3	Participant 12	Developer 2	4
3	Participant 13	Developer 2	2

Table 5.1 List of Participants

From the beginning of selective data collection phase (Phase 3), the researcher took a draft diagram of the emerging theory to the interviews. At this stage, the interviews focused on a few follow-up questions from the previous interviews in addition to specific questions centred around the emerging core category – Meeting User Expectations. Further questions focused on the two overarching themes emerging from the data: challenges and innovations to overcome the challenges in light of the fact that Scrum does not prescribe any techniques or rules for implementation of SQA processes. After the initial questioning, the researcher would introduce the interviewee to what has emerged so far and would let them say more or query some of the ideas captured in the mock up theory. Keeping the interviewing process this way enabled room for comments and differences in opinion which reflected differences in how they experienced various SQA practices. It also sharpened thinking around the naming and the definition of the main categories.

During Phase 2 and Phase 3 the researcher also observed daily Scrum meetings. The most important thing noted from the observations was the nature of collective effort, knowledge sharing, and mentoring. Transcribing the interviews was a laborious and time consuming

exercise even though the immersion in data proved invaluable throughout the constant comparison process.

5.2 CANONS OF CLASSIC GTM

5.2.1 Data Incidents, Concepts, and Categories

Data analysis in GTM focuses on generating categories and their concepts. The concepts consists of initial codes which are labels assigned to data incidents. The incidents which are sometimes called indicators, are often captured in the words of participants (Adolph et al., 2011, p. 494). Glaser uses the terms "concepts" and "categories" interchangeably, but to avoid confusion, the researcher chose to follow (Adolph et al., 2012) by thinking of concepts as the basic building blocks of theory and categories as a grouping of concepts. "Concepts are often behaviours or factors affecting behaviours, which help to explain to the analyst how the basic problem is resolved or processed" (Adolph, Hall, & Kruchten, 2011, p. 493). Categories are therefore concepts at higher level of abstraction.

5.2.2 Constant Comparative Method and Rigour in Classic GTM

Constant comparative method entails systematic and simultaneous collection and analysis of data. Constant comparison is seen as a core strategy for producing a grounded theory (Matavire & Brown, 2011). This strategy is based on a concept-indicator model in Figure 5.1 (Adolph et al., 2011) and involves comparing at three different levels: The first level involves comparing data incidents to data incidents to clarify uniformity (Glaser & Holton, 2004). Incidents labelled under the same concept are compared to ensure to fit and workability (Urquhart et al., 2010). It involves constantly comparing and contrasting incidents as they emerge from the data to distil similarities, differences and degrees of consistency. The second level compares data incidents to generated concepts to establish theoretical properties of generated concepts. The third level comparisons focus on comparing concepts to concepts.

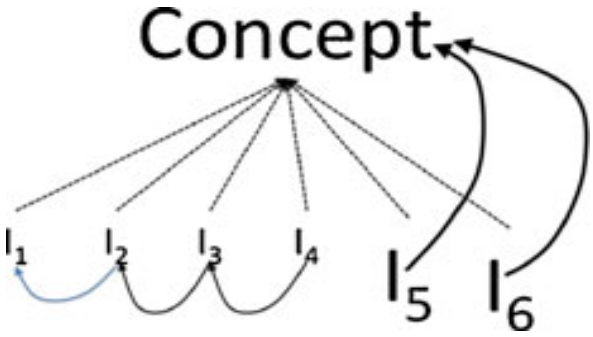


Figure 5-2 Concept indicator model comparing subsequent incidents with concept (Adolph et al., 2011)

Table 5.2 presents the emergence of an example concept (*Capacity Constraints*) and indicators compared and aggregated into the concept. A full list of all indicators and data fragments for all concepts in the final theoretical framework is listed in Appendix B. Capacity Constraints is one of the key issues that came up as a result of the absence of dedicated testers. It relates to knowledge, skill, and time demands placed on business analysts and developers for them to be good at their respective disciplines and also be good at testing. Comparison between the indicators that made up Capacity Constraints concept reveals that the capacity constraints range from analysis work to development work. Business Analysts in a team that does not have Dedicated Testers are required to facilitate and oversee the whole testing process while also taking care of requirements analysis.

Further comparison reveals that developers are also required to write code, to test, and fix the bugs and this leads to *Workload Overheads*. This also means that they are required to be good developers and also be good testers.

The resulting dimensions for the Capacity Constraints concept are:

- **Capacity**: The ability to fulfil both analysis role and a tester's role
- **Pressure**: The amount of work and deadline pressures. When deadlines are tight, testing is always the first to be pushed aside.
- **Expertise**: The need for developers to be good programmers as well as being good testers. This applies to business analysts as well.

Concept	Indicators	Data fragments
Capacity Constraints	Multiple pressures	*"Where we trying to work in this area of multiple, uhm, I suppose pressures"*
	Increased Capacity Demands	*"so at one point in time, that person is going to have a dual role at the same time, which means there is going to be increased pressure on capacity required for that person"*
	Duality of Roles	*"so you are asking an analyst to do an analyst job but also to be facilitating a software testers role"*
	Workload Overheads	*"Obviously with all the testing done by the team there is an overhead."*
	Required Expertise *(lack of)*	*"Developers are traditionally not good Testers"*

Table 5.2 Concept-Indicator model and emergence of Capacity Constraints

A full list of all indicators and data fragments for all concepts in the final theoretical framework is listed in Appendix B. The Data Analysis section in this chapter provides more detail on the coding process.

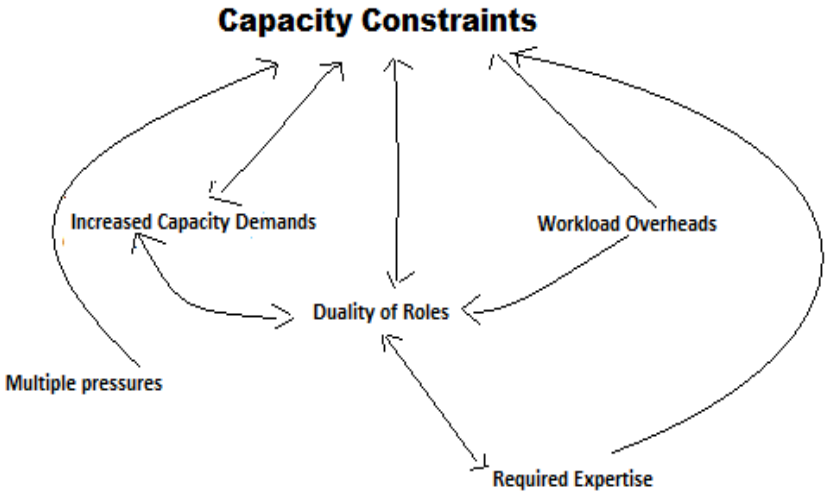


Figure 5-3 Constant comparisons for Capacity Constraints

To make the first level comparisons more explicit, the researcher coded at both sentence and paragraph level. Coding at the sentence level improved immersion into the data and this made comparisons between incidents in different interviews easy. Another factor that contributed to making this process possible was the transcription of recorded interviews. The researcher used both Microsoft Word and Microsoft Excel to perform comparative analysis. Microsoft Word was used for initial coding and the codes together with memos for each code were written as 'Comments' on the right panel. After initial coding, the researcher transferred the initial codes, the data excerpts, and the memos into Microsoft Excel columns for further comparative analysis (see Appendix B). The initial codes were kept as descriptive as possible. During the process of comparing incidents, the researcher constantly reverted to memos to regain reasoning, to challenge and refine own thinking. Using Microsoft Excel, the researcher was able to filter through similar codes and do comparisons. At later stages of the study, the researcher would filter by concepts to compare all initial codes under the concept, or filter by categories to compare all concepts under the categories and their initial codes do further refining.

There is no grounded theory without constant comparison (Fernández, 2004). This process forces a researcher to think widely about a category by considering its dimensions, conditions

that minimise or pronounce it, its consequences, and how the category relates to other categories and to its properties (Glaser, 1965). It improves consistency and minimises bias. It also refines definitions given to concepts by constantly comparing and renaming data incidents and concepts. The aim is to ensure that the generated concepts fit the data and are workable (Urquhart et al., 2010). This process is a key component of GTM's procedures for ensuring rigour in order to discover theories that enhance understanding, explanation, and prediction of phenomena under investigation (Lehmann, 2010).

Details of the coding process through the three levels of coding are discussed in the Data Analysis section.

5.2.3 Theoretical Sampling

Theoretical sampling is the process through which a researcher jointly collects, codes, analyses the data, and uses the analysis to decide on what data in needed and where to find the data based on the emerging theory (Adolph et al., 2011) . It refers to "the process whereby the researcher selectively samples the next data to collect whilst jointly collecting, coding and analysing the data" (Roode & Niekerk, 2009, p. 101). The properties, dimensions, and variations of generated categories dictate theoretical sampling (Strauss & Corbin, 1990). As a researcher moves from one interview to the next, he/she is guided by the concepts that emerged from the previous round, and it is important that he/she watches for reoccurrence of important concepts from previous interviews in the next batches of data. This process cannot be pre-determined because it is driven by the emerging theory. It differs from purposeful sampling in that, it is directed by the emerging theory (Matavire & Brown, 2011).

The researcher sought comparative incidents through theoretical sampling with the sole purpose of generating categories and their properties drawing inspiration from Glaser and Holton (2004). For example, Participant 2 mentioned that although the team did code reviews, he still believes that the level of quality is not good enough. He had also complained about the quality of testing from developers. He further mentioned that he believes the quality is not so good because the developers are subjected to pressure to finish as many stories as possible

within a sprint. He then went to suggest that the reason for the scrapping off of one release was because of poor quality.

Based on the analysis of the data from Participant 2, the researcher had to find out more about the mechanisms put in place for ensuring code quality, the experiences of developers with testing, the focus on quantity of stories, and why there were no dedicated testers. The researcher therefore saw it fit to look for a more senior Participant, who would be in a good position to provide information about quality control mechanisms, sprint planning decisions, and hiring of personnel at SAIT. For this, Participant 3 - an Application Architect - was identified as the most senior member of the development team and thus the right candidate for the third interview. He stressed the importance of '*developer testing'* and '*code reviews'*. He mentioned that developer testing as part of the '*process workflow'* was new to SAIT. He mentioned that their approach had evolved and that their estimation process had improved. He however bemoaned the '*workload overheads'* as a result of team doing all the testing. He further alluded to '*time constraints'* and that regression tests are therefore not given enough time.

Taking the issue of quality further, Participant 4 - Senior Project Manager - mentioned the need for a Scrum process to be supported by techniques from other Methodologies such as XP. Following the same principles of theoretical sampling, the researcher learned that there are pros and cons for doing code reviews when Participant 5 mentioned the '*diminishing value of Code Reviews*'. He stressed the importance of Code Reviews for improving the learning curve for new starters and making sure that coding standards are adhered to. But he mentioned that sometimes Code Reviews can lead to unwarranted arguments over semantics when senior developers review each other's code. The choice for the level of experience and role for the next participant was always guided by the gaps in emerging theory.

5.2.4 Theoretical Memoing

Memos form a fundamental element of GTM (Strauss & Corbin, 1990). It records theoretical ideas about the data and the emergent theory (Roode & Niekerk, 2009). The process of keeping memos enables one to track all categories, properties, conceptual relationships, and generative questions as they are discovered from the data. This process begins at the first coding session

until the research is completed. The researcher recorded memos to outline ideas about interview sessions and generated categories. Other memos were arbitrary questions and ideas about whether one incident indeed belonged to a certain high level category. It is important to confess that the memos were not always nicely organised. This is due to the fact that the best ideas seemed to strike in unconventional settings and the researcher would grab piece of pen and paper and jot down the ideas. These neatly written memos allowed for a place to jot down areas for further exploration, own internal *debates* on abstraction of codes, and confusions around where certain codes belonged, and reasoning around coding process. Appendix C and Appendix D show example memos.

5.3 DATA ANALYSIS

Classic GTM coding techniques were employed for data analysis. After transcription of each interview, the researcher would engage in initial coding, memoing, and comparison of interview data incidents with previous incidents. The aim of coding is to generate categories and their properties. As several codes were discovered from raw data, those that were deemed related were grouped into higher level categories (Strauss & Corbin, 1990). The unit of analysis was the organisation, the project, and the development team. It was therefore important to look at the data from a holistic level while also grounding codes by coding at a sentence level.

The researcher focused on increasing the level of abstraction as the research process evolved through the coding steps. In addition to raising the level of abstraction, "theory building requires abstraction to classes that cut across different empirical environments and transcend to nomothetic constructs that are independent of the actual units of enquiry" (Lehmann, 2010, p. 3). These basic principles were executed extensively. For example, '*Testing Coordination'* was abstracted to '*Work Coordination'*. Various individual SQA techniques from other software development methodologies such '*Test Driven Development'* were abstracted to '*Adopted Practices'*. These basic principles were also helpful in deciding the core category.

More often than not the researcher found naming concepts and categories tricky. The main problem was trying to come up with a best-fit name. According to Glaser as cited in (Adolph et al., 2011, p. 495) validity in GTM is achieved after "much fitting of words, when the chosen one

best represents the pattern". Details of how each of the three coding steps in Classic GTM were employed in this study are elaborated in the next sections.

5.3.1 Open Coding

Open coding involves naming incidents, making comparisons between incidents, and recording memos (Roode & Niekerk, 2009; Urquhart, 2001; Fernández, 2004). Each data incident is coded into as many categories as possible. Open coding was always the first coding done for each interview. Coding at a sentence level ensured that the generated categories were as close to the data as possible. This stage had many inconsistencies at the beginning of the research process. Some incidents were coded as low level descriptions whereas others were coded at higher abstract level. After further reading on coding, particularly in the IS field, the researcher had to re-do initial coding for the first three interviews.

Some categories tended to saturate quickly, for example, the Adopted Practices category saturated quickly as all respondents mentioned code reviews, test driven development, and developer testing in way that did not add any variation to the category. This category saturated after the sixth interview. Other categories took longest to saturate, for example, Capacity Constraints was the longest lasting. The reason being that most participants varied in opinion on whether it was good or not to have analysts and developers do the testing. The differences were around expertise as some participants believed that further training can enhance developer/analyst testing whereas others believed that developers or analysts were simply not motivated to be good testers and that testing is a skill in its own right and needs someone who has it as a profession.

5.3.2 Selective Coding

"Selective coding means to cease open coding and to delimit coding to only those variables that relate to the core variable in sufficiently significant ways as to produce a parsimonious" (Glaser, 2004, p. 11). Selective coding started after analysis of the eighth interview when the core category emerged through constant comparison. Deciding on the core category took a lot of thinking and back and forth analysis. The researcher waited for the core category to emerge and ensured that the core category was only selected when there was no other category that

seemed to account for all other categories more than this category. During the early stages of the research process, the researcher was tempted to select 'Dedicated Testing' as a core category. This category seemed to account for all challenges around SQA in the case organisation. It also seemed to be the key concern from all participants. But the researcher had to 'wait' and consistently question the data, apply creativity, and think at level higher than the data. Glaser (2004, p. 12) advises patience and that the analyst "must accept nothing until something happens, as it surely does".

Dedicated Testing as a core category was not enough as this category was linked to the fact that there was no dedicated tester at the case organisation. Dedicated Testing as a core category would not cut across empirical environments. Another contender for core category was 'lack of prescriptions'. The issue around lack of prescriptions had been present in all interviews and the researcher could not find a suitable term or phrase to use to capture the issue. It was only after the eighth formal interview that the researcher decided to do some literature review to find appropriate terms to define this category.

It was also important at this point for researcher to really compare with the original Scrum monographs whether Scrum does not provide prescriptions on SQA strategies. One article on agile methods (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003), confirmed that in fact Scrum does not offer prescriptions, and the category – Concrete Guidance, was born. It is important to note that according to Glaser (2004) the literature can be treated as another source of data, can be used in constant comparative analysis, and can improve theoretical sensitivity.

The need for Dedicated Testing and Concrete Guidance did not account for all variation in the data on their own. Another category 'Need for Solid User Representation' emerged as a strong contender for core. But this category on its own also did not account for all variations although it seemed to be the essence of all mechanisms put in place for achieving user quality requirements. Digging further into the data, the researcher discovered that there should be a reason why the participants sought solid user representation and that there should be some concern being addressed by the different innovations put in place as a result of no concrete guidance in Scrum. Further, the researcher studied the challenges and probed the data to find

the concern which leads to participants calling for dedicated testing and encountering the challenges as a result of lack of dedicated testing. This questioning, this interrogation of the data, and this relentless comparison of the concepts and indicators within all categories lead the researcher to the core concern: Meeting User Expectations. This core category embodies all three contenders (Need for Dedicated Testing, Need for Concrete Guidance, and Need for Solid User Representation) as its concepts.

The researcher decided it was important to go back to the analysis of the interviews in an effort to ensure that all aspects related to the core category were captured. Following Urquhart et al. (2010, p. 362), "the saturated concepts were reduced as much as possible to the relationships between the core categories". In addition to coding selectively, the researcher engaged in further selective data collection with the prime focus being around the core category. The aim was to elaborate the relationship between the core variable with other categories.

5.3.3 Theoretical Coding

The aim of theoretical coding is to stipulate explicit relationships amongst the categories (Urquhart et al., 2010). This stage of coding is critical in producing theories (Glaser & Holton, 2004). The relationships between categories can be stated in a form of associations, influences, or can be causal (Urquhart et al., 2010). Recording memos extensively at this stage was important to keep record of thinking as the researcher sought linkages between categories. The researcher relied on drawing diagrams and writing mock stories of the enfolding theory. This step in coding epitomises the essence of iterative conceptualisation and provides an opportunity to think creatively about the data.

The GTM researcher has a range of theoretical coding families to use for conceptualising how categories relate to each other (Hoda et al., 2012). In this study, applying theoretical coding towards the end of analysing resulted in some categories being conceptualised into Challenges and Innovations as overarching categories. Within the Challenges, further theoretical coding illuminated that Capacity Constraints, Lack of Testing Expertise, Testing and Quality Issues were ***consequences*** of the absence of dedicated testers. It also emerged that Developer Testing, and Process Workflow were ***elements*** of the Process Structure category.

It is important to note that the researcher did not focus on causality between the categories as this is limiting in interpretive research (Urquhart et al., 2010). This does not mean causality was discarded. The researcher only remained open to the emerging theory. According to Tan (2010), a GTM researcher must be creative and must question, reason, and make sense of the relationships between categories. These cognitive processes require a great deal of memoing. Contrary to many academic activities, the researcher found that a majority of theorising was done in informal and unconventional settings, such as in a train to work, in a bar, or in bed during one of the many sleepless nights. One of the most productive attempts at theoretical coding was done after a Friday night out with friends! This attempt was also a mock write-up of the theory.

5.4 SCALING UP

5.4.1 Theoretical Saturation

Data collection proceeded and was informed by theoretical sampling decisions until no new concepts emerged. After the emergence of the core category, the researcher continued collecting data with a specific focus on the concepts of the core category. The researcher continued gathering more data even after it was clear that no new significant concepts were emerging. This was an attempt to remain open for any new relationships, contradicting evidence, or further densification of established categories. At some point, the researcher needed to 'trust' the data and the established relationships. At this point the memos were revisited and the mock storyline and did further sorting and rework.

5.4.2 Delimiting the theory

After reaching saturation, it was important to scale up the theory and decide on the story line. To achieve this, the researcher needed to 'leave' the data and think about the results of theoretical coding. Urquhart et al. (2010) state that grouping high-level categories into larger, broader themes is a successful mechanism to use in order to rise above the detail and consider the bigger picture. The researcher noted the importance of rising above the generated categories to give them a substantive meaning. This exercise produced two overarching categories: Innovations and Challenges in Meeting User Expectations. The essence of this is that

due to the lack of concrete guidance in Scrum, a development team has to devise innovations such as adopting practices from other methodologies in order to meet user expectations. Further, the absence of dedicated testing in a Scrum environment poses challenges such as capacity constraints, testing issues, and quality issues. A detailed elaboration of these is given in the Findings chapter which follows this chapter.

5.4.3 Sorting

Sorting involves integrating different memos related to the core category and its properties (Glaser & Holton, 2004). This stage requires some thinking about the data, the memos, the generated concepts, and the relationships. According to Glaser (2004, p. 7), "relevant theoretical codes emerge in conceptual memo sorting and could be whatever". In order accomplish this, the researcher listed all major categories related to the core category on a paper sheet. Then, the researcher started writing up on these categories without looking at the memos or the data. Doing this exercise made it possible to determine whether the generated categories made sense and were simple and yet grounded to the data. The researcher then visited the memos to make sure that the write up was inclusive of all major ideas. Refer to Appendix D for a detailed real-time memoing, sorting, and theorisation.

Later on this mock write-up became a central point for further selective coding, constant comparison, and theoretical memoing. This is where major theoretical categories as they appear in the final theory were generated. Work Coordination, Adopted Practices, and Process Structure were derived after initial sorting. Reading the mock write-up also allowed for further writing down of memos and refining of categories. This also allowed for refinement of the significance of concepts such as Capacity Constraints, Testing Expertise, Testing Issues, and Quality Issues. For example, Capacity Constraints was initially an independent high level category. After the first attempt at sorting, and further thinking, it was clear that this category was actually a consequence of the absence of a dedicated tester.

It is important to confess that some concepts under the Adopted Practices category, such as Test Driven Development, and their importance on SQA in an Agile environment might have been pre-conceived. The minor literature review highlighted the importance of TDD in Agile

developments. This might have influenced the decision to include these as concepts under the category. However, it is important to note that these concepts were present and dominant in the data, either through interviews, ad hoc conversations, software development team meetings, and introductory presentations given to new graduates. Code Reviews and Collective Ownership were emphasised during induction presentations when the researcher first joined SAIT. However, it is also important to note that some pre-conceived practices such as the "System Metaphor", "Agile Model Driven Development", and "Pair Programming" do not feature in the theory at all because they were never emphasised by the participants.

According to Glaser (2004), sorting results in theoretical completeness, and provides internal integration between the categories. It also affords the researcher, an opportunity to decide on the relevance and significance of categories. The next Chapter focuses on the substantive theory, its categories, the concepts, and the relationships between the categories.

CHAPTER 6. FINDINGS

6.1 MEETING USER EXPECTATIONS

The material presented in this chapter is the outcome of analysis of raw data analysed through Classic GTM techniques. This chapter does not provide an integrated analysis of the findings with the literature. It presents the findings integrated in a theoretical framework of Figure 6-1 which depicts main categories and their concepts. The categories are integrated by a core category which embodies the core concern on the aspects of software quality assurance in a scrum environment. Chapter 7 provides an interweaving of the findings presented in the chapter and the literature

The core concern at SAIT is "Meeting User Expectations". This concern was evident in all interviews. Some participants mentioned it explicitly while others did not state it explicitly although it was evident in the texts. In addressing the concern, the participants alluded to the three concepts: "Need for Solid User Representation", "Need for Concrete Guidance", and "Need for Dedicated Testing". As depicted in Figure 6-1, the lack of dedicated testers at SAIT presents challenges to meeting user expectations and the lack of concrete guidance in Scrum requires the project team to come up innovations in order overcome the challenges and improve the ability to meet user expectations. It is also important to note that some of the innovations were devised to minimise the impact of the absence of a dedicated tester and to improve solid user representation.

The lack of concrete guidance concept is central to almost all categories. For example, solid user representation is easily achievable when there are concrete prescriptions on how to handle user requirements elicitation and articulation. Similarly validation and verification processes require concrete guidance. In the same manner, software testing requires concrete guidance on how to plan, write test cases, and execute different testing activities. In the case of testing, the absence of a dedicated tester at SAIT worsens the challenges for the Portal Project team.

The theoretical framework of Figure 6-1 proposes understanding Scrum as a framework of "empty buckets" which needs to be filled with situation-appropriate processes, techniques, and practices in order to meet user expectations. The empty buckets analogy came up during one of the interviews as one participant emphasised the absence of concrete guidance in Scrum. All senior participants (*more than 5 years experience*) pointed out the absence of concrete guidance as a big concern. This concept accounts for a majority of the innovations and some of the challenges such as the Testing Issues and Quality Issues. This concept is more related to the innovations such as Adopted Practices and the design of the Process Structure.

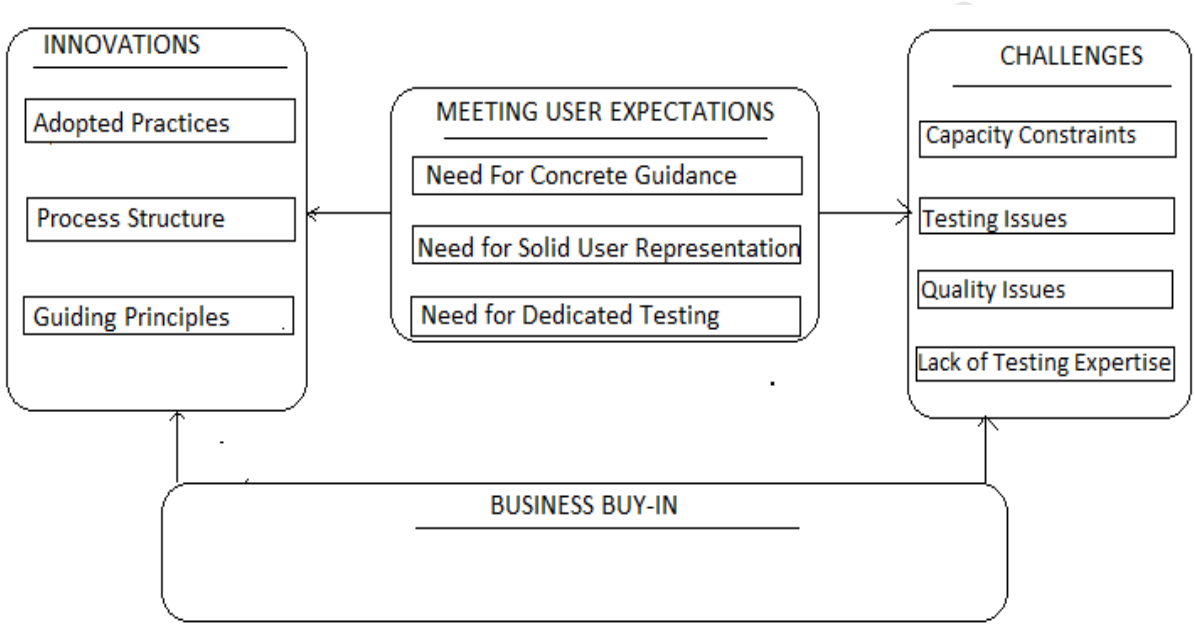


Figure 6-1 Aspects of Software Quality Assurance in a Scrum environment

It is important to note that most participants attributed the challenges to the lack of dedicated testers at SAIT. When quizzed why there were no dedicated testers, senior participants pointed to two reasons; Lack of business buy-in and the fact that Scrum does not explicitly prescribe how dedicated testers fit in into a development team. Junior participants mentioned that the Portal Project started off as an in house 'play-ground' project for Java developers who were not resourced to client project and as such there was no need to employ a dedicated tester.

The lack of concrete guidance in Scrum and the absence of dedicated testers at SAIT hampers the Portal Project team's ability to meet user expectations. More importantly, the team finds it

hard to achieve required solid user representation as the idea of a full-scale user involvement is unrealistic. Solid user representation is particularly pressing at SAIT because the absence of a dedicated tester makes it challenging for the team to confidently ascertain that user stories accurately reflect user needs and the finished product features adhere to the user needs.

The next sections provide a more detailed elaboration of the concepts and how they relate to each other. The presentation focuses more on the 'Need for Concrete Guidance' and the 'Need for Dedicated Testing' concepts as the researcher deemed these two concepts most important to SAIT and to a Scrum environment and are an interesting part of the theory. User representation covers aspects around user requirements elicitation, analysis, articulation, verification, and validation. The Information Systems literature provides a lot of insights into these aspects and as such this study does not provide a detailed account of these aspects.

It is also important to note that the presentation in this chapter is not a linear textual version of the theoretical framework. For example, the section that focuses on the Need for Concrete Guidance is not linearly followed by a section for discussing the 'Need for Dedicated Testing'. Rather, Concrete Guidance is presented and interweaved with the Innovations because the lack of concrete guidance requires a development team to devise the best processes, techniques, and practices for meeting user expectations. In the same way, the lack of dedicated testers has had the many consequential challenges and it is therefore fitting to discuss the Need for Dedicated Testing together with challenges.

6.2 CONCRETE GUIDANCE

6.2.1 Need for Concrete Guidance

One of the main concepts within the core category relates to a lack of concrete guidance on how to incorporate and implement SQA processes and techniques in Scrum. Participants at SAIT stated that, Scrum does not provide any prescriptions on how to design the process structure, it offers no guidance on how to coordinate SQA activities, and does not define what specialised disciplines and skill sets are necessary when composing a team. The study has identified that this provides room for innovation based on the situation at hand.

The lack of concrete prescriptions came up in the first data collection meeting with SAIT's Head of Software Operations, the Service Desk Manager, and the project team's ScrumMaster. In that very first meeting, all of them highlighted the fact that Scrum is a management framework, and does not prescribe any rules, any guidelines, or any processes that a team should use for SQA.

Participant 4:

"*Scrum as a methodology does not have any rules around quality*"

This aspect about Scrum came up in all subsequent interviews. All participants in the eight interviews (*formal recorded interviews of data collection phase 2*) alluded to the fact that Scrum does not prescribe any standards to follow, it does not offer any guidelines on how to actually do the work, it does not offer a definition of "done", and leaves everything open to the team. The onus is on team members to deliver a quality output at the end of the day.

Participant 1:

"*We are unknowingly applying SQA, because we know that it needs to work at the end of the day*"

Participant 2:

"*I don't think we even know what we supposed to be working off as the base*"

The innovations that the Portal Project team had to devise include designing the process structure, adopting SQA practices from other methodologies, and using principles of collective ownership, constant feedback, and continuous improvement while doing devising the innovations. As Participant 8 stated, allowing a development team to innovate provides empowerment, sense of ownership, and flexibility to adapt to different situations. These aspects are discussed in the next sections.

6.2.2 Process Structure

The Process Structure category denotes the processes, phases, techniques, and responsibilities to transform requirements on the Product Backlog to potentially shippable piece of software. Essentially, Process Structure embodies a team's "Definition of Done" on a story and in this case includes the activities, the roles, and the tools. For the Portal Project team, the structure of the process is designed in a way that allows for incorporation of practices and techniques from other software methodologies. These practices are; Developer Testing, Code Reviews, and Continuous Integration. At the end of this study, one lead developer (Participant 11) had completed a major testing strategy Research and Development work to determine how to integrate Test Driven Development in the Portal Project process structure. In addition to adopting practices, process structure is also concerned with the process workflow and coordination of various activities and work dependencies.

The concepts within the Process Structure category are: Process Workflow, Work Coordination, and Situation Appropriateness. These concepts are discussed in the next sections.

Process Workflow

The main concept within the process structure category is Process Workflow. All senior participants advised on careful design of the process workflow to incorporate practices aimed at meeting user expectations. According to the participants, Scrum does not offer any concrete advice on how to design the workflow. It leaves everything open for development teams to figure out. According to the participants, the process workflow *can* consist of just three phases: "Ready", "In Progress", and "Done". When the process workflow has only these three phases, it becomes difficult to incorporate SQA practices within the process structure. Alternatively, it can include many sub phases between "Ready" and "Done". The Portal Project team's workflow has "Ready", "In Progress", "Ready for Test", "In Testing" and "Done". Participants believe that carefully designing the structure can significantly help meet user expectations.

Participant 5:

"So what I would recommend for a new team that is starting is to really think about that workflow for a bit and understand how you can incorporate testing into it."

The design of this workflow is fundamental to ensuring delivery of software that meets user expectations. It determines and affects how a team approaches and organises its work. For example, when a story is moved into "Ready for Test", it means one developer has to pick it up and first do a code review and thereafter do the testing. This aspect also demonstrates the concept Collective Ownership in the Guiding Principles category while also touching on the Adopted Practices category.

Participant 3:

"So this whole process that I described where the code reviews and functional testing are built into the workflow has worked well for us."

These aspects demonstrate the relationship between the Process Structure category and the Adopted Practices category in that Developer Testing and Code Reviews as Adopted Practices are mandatory within the process workflow. According to all participants, Scrum does not say anything about these software engineering practices and does not offer any guidance on how to incorporate them activities.

One of the phases that has to be built in the process workflow is testing. Participant 5 stated that it is important that a development team does not assume that testing will happen outside the workflow. Coordination of the work between analysts, user experience developers, and the programmers is one area that requires careful innovation, particularly when it comes to testing.

Work Coordination

Work coordination is about how the processes and roles within the process structure should be organised and how the different roles should interact in the delivery cycle to collectively work towards meeting user expectations and delivering quality code. This concept simply denotes dependency resolution within the process structure. It is concerned with ensuring that input artefacts are available as and when the consumer actors need them. For example, participants mentioned that it is important that user interface screens and wireframes are ready before developers start writing code for the screens. It is also important that analysis work on user

stories is complete before developers can start. These aspects require tight coordination and dependency resolution mechanisms.

One critical aspect at SAIT in the Project Portal team is around testing coordination. There is a lot of work involved in planning for testing which includes setting test environment, organising test data, and so on. This means that coordinating testing requires a lot of thinking from the leadership team. This has led to some participants arguing for need more prescriptions on how actually organise testing work.

Participant 5 (the ScrumMaster) concluded by saying the following:

"I personally would like to come across a material that gives me a testing strategy that I can readily expect to work

Participant 5

"There is nothing in Scrum that says well this is Scrum for when you taking the product to the market and this is Scrum for when you already have a product in the market."

Dependency resolution for analysis and design work is not a big problem for the Portal Project. The analysts ensure that they are one or two sprints ahead in terms of analysis work. This means that in current sprint, the software analyst and the product owner begin writing stories and other related activities for work that will be done in the next sprint. In a similar manner, user experience work commences one or two sprints ahead which means that user experience developers start working on user interfaces for features that will be part of work to be developed in the next sprint. This innovation was referred to as the "N+1" concept by the ScrumMaster (Participant 5). The coordination of analysis work and user experience work is one example innovation that is working smoothly at for the Portal Project team.

On the other hand, testing work requires a little more innovation over the basic N+1 concept. According to the ScrumMaster, there are at least three important aspects to testing that make applying the N+1 concept difficult in testing. Firstly, testing can only begin when there is work ready for test. This means that at least one story has be completed by developers. Secondly,

there will be work that is completed on the last day of the sprint that needs to be tested before being released to the client. Finally, test cases, test data, test environment need to be ready before testing can begin.

While the N+1 innovation works smoothly for analysis work and user experience, the exact nature of how dedicated testers are supposed to be involved in a two week sprint is not clearly defined at SAIT. The main question relates to the timing for testing work that is completed on the last day of Sprint. Through the N+1 concept, testers can start preparing test cases for the next sprint during the current sprint while at the same time being available to test new features as they get developed in the current sprint. However, there will be some work that will only be completed in the last day of the sprint that still needs to get tested before release. One option is to test this work in the next sprint. In that case, the Tester does not only work N+1. The tester does work for the last sprint (N-1), the current sprint (N), and the next sprint (N+1). The exact nature of how this can work is not yet fully understood and requires more creativity.

The one option that is currently in place in the Portal Project team is the concept of a "Doctor". At the beginning of each sprint, the team selects one team member to be the "Doctor". The doctor is on 'stand-by' for the whole sprint to work on any bugs that are found on work released in previous sprints. This concept is working well in light of the fact that there are no dedicated Testers and no dedicated testing phase at SAIT.

Situation Appropriateness

Almost all participants stated that the exact nature of how work should be organised and what practices to adopt including team composition depend on the nature of a project. They supported this by saying that what works well in one environment may not necessarily work well in another environment. For example, what works well within an internal project may not work for a client project. An internal project may have a strong product owner with a clear vision and power as in the case of the Portal Project. On the other hand, a client project may not have the same support and vision.

Participant 7

"When you choose how to structure your team, and when you choose how to structure your approach or project, I think it is very dependent on the project"

As such the Scrum team will have to innovate how to organise and coordinate its work based on structural constraints. Structural constraints can include the source of user requirements, the availability of testing environment, and the availability and location of the "customer".

Some participants argued against asking for more prescriptions by advising that people have to understand that Scrum is a management framework and it leaves room for situation-based innovation.

Participant 8

"You know there is a lack of misunderstanding out there. Scrum is a framework of empty buckets, it gives you a backbone and you need to fill the buckets"

The lack on concrete guidance and prescriptions on how to coordinate development can pose challenges and at the same provide room for innovation. The most important part is that Scrum stakeholders need to understand that Scrum needs to be built around situation appropriate innovations. The next section presents practices adopted by the Portal Project team and those that are in the pipe-line for adoption.

6.2.3 Adopted Practices

The study revealed that the process structure must be designed in such a way that practices from other software development methodologies such as XP can be incorporated. Participant 4, who is a devout Scrum proponent promoting Scrum at SAIT mentioned that Scrum helps a team code faster but it needs right control mechanisms to produce good quality code. According to the Participant, if the project team does not adopt right quality control practices then they can write poor quality code quickly.

Participant 4:

"...but if you do not have right control mechanisms in place, you can code bad stuff quickly, it's all going to happen quickly."

As such, the process structure should make it possible to adopt Software Quality Assurance practices and build them into the release cycles. Such important practices include peer code reviews, developer testing, and test driven development. The next sections discuss these practices in detail

Developer Testing

In the Portal Project, developer testing as part of the process workflow means developers test each other's work before the work can be marked as done. The story has to first be marked as "Ready for test". "Ready for test" – as one of the steps in the workflow - means that another developer needs to first do a code review. At the completion of the code review process, the story will be moved to "In Testing" whereby another developer will do a functional test. This means that every piece of functionality has to go through a code review process and developer testing by the development team.

Participant 3:

"Obviously the quality has been better since we moved to Scrum, because previously we didn't have this predefined necessity to test every story before it makes it into production release"

Although developer testing plays a big role in meeting user expectations, some participants expressed concerns over this. For example, Participant 3, the Application Architect in the team, expressed that there is a workload overhead as a result of all testing being done by developers. There are differing views to this as three other senior participants mentioned that it is important that the team takes responsibility of their code and ensure that it works before passing it on to other people.

Participant 8:

"to me you are as good a developer as you are a tester..."

Code Reviews

As mentioned in the Developer Testing section, every piece of work that gets addressed by the team has to be peer reviewed before it can be tested. The Portal Project team uses code reviews to ascertain that the written code adheres to stated coding standards. Code reviews present an opportunity for team members to continually improve their code base and share learned lessons.

Participant 4:

"....you generally do have peer code reviews, so that also improves the quality because you have got more than one person doing it"

All participants mentioned peer code reviews as being integral to the software development process. Participant 3 mentioned that although developers are not traditionally good Testers,

"... Peer reviews and peer testing does bring out a number of bugs".

Participant 2:

"The team ensures that a story is of the highest quality because somebody else has refactored on whether I have achieved what the business wants".

Although code reviews are good for ensuring code quality and helping junior developers improve their coding skills, there are some negative aspects to them. For example, two participants felt that some team members impose their own coding preferences on other developers. This results in a lot of arguments over semantics and wastes time.

Participant 5:

"Arguments over semantics after code review feedback happen all the time"

Participant 5 talked about what he referred to as "diminishing value of code reviews". This phenomenon occurs when two seniors developers review each other's code. They waste of a

lot of time that could be used for productive development arguing over semantics, standards, and so on. In most cases when such arguments ensue, one finds that the two arguing seniors developers are “almost always right, just that one is more correct than the other”.

The importance of code reviews therefore depends on the makeup of the team. When a team has junior developers it is important that their code is reviewed for rapid learning.

Test Driven Development

All developers who participated in this study, formally or informally would like to see Test Driven Development (TDD) in place at SAIT, particularly in the Portal Project. When asked on what practices the team should incorporate in the process structure, Participant 4 had this to say:

“TDD definitely!”

It should be noted that the Portal Project team has not practiced Test Driven Development. It is one of the processes that the team has learned through experience that they need to be incorporate the practice. At the end of this study, Participant 11 had just presented his findings on how to incorporate TDD in the Portal Project.

Participant 3:

“We have not done TDD, it is something that we fantasise about...I would really love to try them out, but there is a reluctance from stakeholders to grant so much time to try retrospectively go and write automated tests for this legacy code...”

This is where the lack of prescriptions or guidance on what development practices to incorporate can affect a team. The point mentioned by the Participant above is lack of business support. The lack of prescriptions on what practices to adopt makes it difficult for development team leadership to motivate for certain practices if they are not mandated by Scrum. This is particularly relevant in a time based organisation. This point is dealt with in more detail under Business Buy-in category.

In essence, the design of Process Structure is important for meeting user expectations in a Scrum environment. As Scrum does not provide any guidance on what practices to adopt, it is important for organisations to carefully include these activities in their process workflow. This is not a challenge, but an opportunity for organisations to tailor their delivery cycles according to the demands of a situation.

6.2.4 Guiding Principles

A closer look at the data reveals underlying principles driving the design of the process structure and adoption of practices from other methodologies in order to meet user expectations. These principles are common in all Agile methodologies and guide the design of process structure and the choice of practices to adopt: Collective Ownership, Constant Feedback, and Continuous Improvement.

Collective Ownership

Four senior participants explicitly stated that collective and collaborative code ownership is important to meeting quality requirements. The belief is that if more than one person work on a feature and the team participates in design and planning sessions, then the quality of the software produced is better. This closely relates to code reviews and developer testing. The whole idea is that, the final output is a result of a collective effort and the team has ensured that they have inspected the code to make sure that it adheres to required standards.

Participant 4:

"A team should ensure that people are following the concept of collaborative group ownership of code and stories and it's not the case of that story is yours, that story is yours, etc"

In addition to code ownership, collaborative design sessions help to ensure that the team meets quality requirements.

Participant 4:

"The other thing is that your planning, your sprint planning 2, which is kind of your design session, is done with the whole team. So by virtue of that, you don't have one person running off alone figuring out how to do this thing. "

In cases where designs are done before the planning 2 meeting, the designer has to defend their ideas during the meeting. This allows the whole team to have contribution to the overall quality of the design. This according to the participants helps avoid having one person as a single point of failure. Participant 5 saw quality as a bi product of collective team effort. He stated that he does not believe that collective design sessions directly contribute to product equality. On the other hand, Participant 6, who has had experiences in several different organisations and development methodologies, did not think collective design sessions were efficient. He mentioned that it is a waste of time for everyone else to learn about a story on which they are not going to do any work.

Participant 6:

"The velocity here is much slower, too many people work on the same thing, I don't think it is efficient"

Constant Feedback

Constant feedback is the key to meeting user expectations in the Portal Project team. Some practices are adopted to allow the team to get feedback sooner. The workflow is designed to allow the team to get feedback sooner. Working in two weeks sprints and short delivery cycles allows for constant feedback.

Participant 4:

"But with Scrum, if you embed testing, and there are various ways you can build it into your release cycles, then obviously you are getting feedback sooner"

Focusing on early feedback and making incremental releases also allows the project decision makers to constantly assess the delivered value against budget. This means that at any point, the key stakeholders can decide that the delivered value is enough if there are any budget constraints.

Participant 8

"I would say because of the constant reviews it means that at any point in the journey, at each cycle, you have something which is shippable, it is possible to say oh well we got this far, and we going to stop here that's it for now"

This principles helps to ensure that the is always kept aware of user expectations and they can know soon if they are not in the right path.

Continuous Improvement

The absence of concrete guidance means that a development team has to continuously look at ways of improving its processes. The importance of continuous improvement in a scrum environment was emphasised by the ScrumMaster. This aspect is also evidenced by the strong R&D portfolio lead by Participant 11 who had just finished research on the testing strategy. At the beginning of the phase 2 of data collection, Participant 3 presented research on code coverage technologies that the Portal Project team have adopted to assess the coverage of their unit test suit. Participant 3 also lauded the importance of developer testing – which is a practice that the team did not start with.

Participant 3

"*Our process has been evolutionary, and where it has evolved for two years has worked well for us. I think it is good at the moment*"

Participant 4:

"*The other thing is that if you are performing scrum properly, then there should be strong emphasis within your team of continuous improvement, and the continuous improvement philosophy is that at any given stage you can always do your job better*."

In essence, adopting good quality control practices can only be realised when there is a strong focus on continuous improvement. Designing the process workflow, incorporating coordination strategies into the process structure also requires focus on continuous improvement. Participant 10 had just attended workshops where the main focus was on how to better implement the N+1 concept.

6.3 DEDICATED TESTING

6.3.1 The Need for (or lack of) Dedicated Testing

Participant 4 mentioned that it is paramount that the team is composed in such a way that there is at least one person from each of the software engineering disciplines who knows what they are doing. However, the Portal Project team does not a dedicated tester. The absence of a dedicated tester and the challenges faced by the team as a result, came out strongly in all interviews.

Participant 1:

"This is not an argument that you have to entertain; I think we do need dedicated Testers."

And

Participant 2:

"If we use it as a pure debate thing, where I sit now, I believe strongly in having SQA as a dedicated function in the environment"

Other participants were more concerned about knowledge retention. They mentioned that because there are no clearly defined rules and guidelines on how to test, and there are no permanent dedicated testers, it is a challenge to retain testing knowledge. Individual developers or analysts accumulate and develop testing skills through experience, but there is a serious gap when they leave the organisation. This results in a lack of quality consistency when someone else fills the testing gap.

While developers and analysts can still meet functional testing requirements to some extent, regression tests, system wide tests, and integration tests require more sophisticated and

dedicated testing resources than the story level functional tests. These require proper testing expertise and can also result in work overload and a loss of production time.

Participant 4:

"When stuff goes from development, to integration environment, and then to production, you will always need a certain amount of testing especially to touch integration points".

Although the adopted practices help, some participants felt that these were not enough. These concerns are discussed under Testing and Quality Issues Section.

Participant 2:

"Developers do code reviews, they do some testing, so does the product owner, but I don't believe that we do thorough enough that we need something that is dedicated."

The next section discusses challenges that result from the absence of dedicated testers, and the resultant lack of testing expertise. The challenges have been categorised into: Lack of Testing Expertise , Capacity Demands, Testing Issues, and Quality Issues.

6.3.2 Challenges due to the absence of dedicated testers

This section presents some of the consequences of the absence of dedicated testers. The study did not attempt to exhaust all possible consequences. Some consequences were dropped through constant comparison. These are consequences that did not appear to have any more support in subsequent interviews and were not to the researcher.

Lack of Testing Expertise

All participants mentioned the lack of testing expertise as the major challenge that is affecting their ability to deliver quality products. The lack of proper testing skills amongst developers and business analysts was mentioned in all interviews.

Participant 4:

"That ties back to what we have discussed before in that we don't know what we are doing, it's a very harsh statement"

Four senior participants mentioned that in addition to testing, dedicated Testers would also provide coaching to the developers.

Participant 8:

"They don't necessarily need to do it themselves; they can coach your guys and teach them how to do it because otherwise it is not going to be done effectively"

Testing requires a fair amount of upfront planning. The first two participants mentioned that they felt this was lacking as a direct consequence of the lack of dedicated testing personnel. Planning upfront involves drawing test cases, verifying and updating specs, and ensuring that different scenarios are included based on client requirements.

Participant 4:

"the idea of testing as a function in our world is a young idea, it comes with many challenges, we don't plan properly for it , no one actually goes and says what amount of work is it, who do we need to talk to, so that is the main challenge"

Participant 3:

"I suppose the problem is we don't have that skill , I mean we don't have a proper skill for it"

According to Participant 8, dedicated testers think from the other side of the coin, would help change the way the team thinks about customers. The participant also emphasised on the need to have the dedicated Tester as part of the team, included in sprint planning meetings and estimation sessions. This would help in ensuring that testing is included adequately when assigning story points. This would in turn ensure that adequate attention is paid to test planning and test execution.

Participant 8:

"*Making a dedicated Tester part of the team would ensure that nobody forgets about testing*"

However, there are different views about the criticality of the apparent lack of testing skills. Some participants felt that it is only a matter of training for the team to acquire the skills. Another participant attributed the apparent lack of testing skills to pure motivation. He

mentioned that developers and business analysts are not measured by their ability to test or find bugs, but by their ability to design and implement great solution. As such, they are not motivated to improve their testing skills.

Testing and Quality Issues

Consequences of the absence of dedicated Testers include a range of testing and quality issues. The study did not attempt to identify them all, but will present a few that came through from the analysis. One of the major concerns on testing is narrow testing. This results from the fact that developers test one story in isolation. As such, they could have broken something else in the process. This functional testing of discrete features is not enough to cater for regression problems.

Participant 1:

"I mean in this particular instance, we only test towards one sticky ready for test , one story ready for test"

In addition to narrow testing, the problem at SAIT is that developer testing is not effective as it should be because the testers wrote the system and know how it works.

Participant 3:

"...because you know what you have written, and you are testing for a predefined output, so you are not necessarily testing to try to break it"

This differs with how users use the system:

Participant 2:

"When you ask other people outside the organisation to use the application, they look at it differently and I believe we need something that is dedicated"

The lack of proper testing expertise and subsequent quality issues delayed a release at one stage in the Portal Project. According to one participant, they had gone through what they believed was the right approach, but the quality wasn't good enough and therefore the product

was not released. The participants cited the fact that testing had not properly covered all the various scenarios based on how the client uses the application.

Participant 3:

"One of the issues that come up is that doing functional testing of these discrete activities is not necessarily doing regression testing"

Capacity Constraints

One of the key issues that came up as a result of the absence of dedicated Testers is capacity constraints. This relates to knowledge, skill, and time demands placed on business analysts and developers to be good at their respective disciplines and also be good at testing. What often happens is that business analysts in particular spread their attention across different areas and end up being under pressure to fulfil deadlines. When this happens, they often postpone testing and end up not doing it. According to Participant 7, testing is always the first thing to be pushed aside when workload pressures demand more capacity than they can offer.

Participant 1:

"I mean the capacity constraints, so you are asking an analyst to do an analyst job but also be facilitating a software tester's role"

Participant 7:

"And analysts should not be the ones that are doing the testing, analysts don't specialise in doing testing, in particular things like regression testing"

Participant 6:

"Developers are traditionally not good Testers"

The absence of dedicated Testers means that developers and analysts are required to design and implement solutions while at the same time being able to test. This goes further for developers as they have to test and in turn fix the bugs.

Participant 3:

"dedicated testers would alleviate the pressure from developers of having to test the issues and fix them themselves"

The result of this is increased pressure on required capacity from team members. They occupy dual roles and are expected to be good at them. According to the first two business analysts, this results in a lack of adequate knowledge on testing technologies, and lack of awareness of developments in testing. The effect according to Participant 7 is that one becomes a "jack of all and a master on none". The lack of testing capabilities as experienced by analysts in particular, can result in poor testing outcomes as Participant 1 put it this way:

"In one sentence, you only going to test what you can test, I mean what you put in is what you are going to get out"

Further, one Participant mentioned that he believes that all testing should be done by a dedicated tester to pick up any usability errors.

Participant 6:

"Java developers are not good testers and I personally think that testing is a skill on its own right"

The absence of a dedicated tester is one the aspects that highlight the importance of business buy-in. The senior leadership at SAIT has been motivating for dedicated testers for a long time. At the time of writing up the findings, work is under way to bring in a senior tester at SAIT who will pioneer a testing function. The next section discusses business buy-in as seen through the lived experiences of the participants at SAIT.

6.4 BUSINESS BUY-IN

A software development team needs top management support to be able to produce good quality software. The Head of Regional Operations (Participant 9), stated that organisations have to give the development team necessary support in terms of resources, time, and strategic direction.

Participant 1:

"I think management need to enable the team to perform at the level they need to perform"

Time in a time-based organisation such as SAIT directly translates into cost to company. The participants at SAIT were well aware of this fact. The next section discusses Time. As mentioned by most participants, time is of critical importance to meeting user expectations in a Scrum environment. For example, TDD requires that a team write unit tests before writing production code. This means developers spend more time on stories than they would without TDD.

Participant 3

"There has been reluctance from stakeholders to grant us enough time to retrospectively write these unit tests"

And Participant 2:

"...that's where the challenge comes in, because we are a time based business, that to me has almost become a driver for everything...."

Furthermore, time pressure in a high pace, productivity-focused environment makes it difficult for a team member to dedicate time to do SQA practices. Regression tests were considered to be a waste of development time. For that reason, they are not done regularly. Scrum ceremonies such as retrospectives and daily stand up meetings are seen to be a waste of development time by some developers, although there are varying views amongst participants.

Participant 2:

"You got people that want to do stuff quickly because they are driven by time"

In addition to time, meeting user expectations requires resource support from management. dedicated testers as has been previously discussed are crucial in a Scrum environment. The team in the case had been asking for a dedicated Tester for a long time without getting one. When asked why there was no dedicated Tester, Participant 8 said:

"That's a good question, but we have managed to win that battle. I think the reason why we don't have or we didn't have a dedicated Tester is that, unfortunately most people look at the short term expense. "

The ‘battle’ was a also a result of a misunderstanding about what Scrum is. The misunderstanding stems from how the idea of Scrum is often communicated to decision makers.

Participant 8:

“Scrum is seen as a silver bullet that is capable of destroying everything which this makes it difficult to convince management to grant you time and resources”

Participant 2:

“I think it is a difficult challenge, they will say yes we all buy in into the notion of having a Tester, it all sounds good whatever the case is, when it come to the cost, we need these dedicated people whatever but it is going to cost X”

Meeting user expectations requires business buy-in otherwise a team will not perform at its best.

6.5 SOLID USER REPRESENTATION

Solid user representation encompasses a broad range of aspects related to user requirements elicitation, analysis, articulation, verification, and validation. The concern embodied in this concept is to ensure that the team clearly understands what the users want and can ascertain before user acceptance testing that they have indeed delivered according to user expectations. This means that user stories should accurately reflect user expectations, and the processes for validation and verification should be solid enough to ensure to ensure that the expectations are met. Solid user representation captures the need for adequate user involvement of which most of the participants mentioned that it was almost impossible to have.

Participant 3:

“*I would change the way the client is seen, so someone like XX really understands what the business wants, so I would just have someone to facilitate that process and put more around deeply understands what the business wants.*”

Participant 2:

"*I think from where we are right now at SAIT, my role from a BSD perspective is making sure that a story that you put down on the product backlog meets user's requirements*."

Participant 4:

"*Ideally your team is working with your users directly so they sit with them, but that is not always realistic.*"

The lack of adequate user involvement requires team members to be the representative and to have a clear understanding of what the user wants. Solid user representation should ascertain that the stories accurately represent what the user wants and that the finished product features are indeed what the user expects to see. Some participants stated that dedicated testers are ideal for achieving solid user representation.

According to Participant 2, dedicated testing would help client-centric thinking within the team, it would improve client representation, and would ensure that there is someone in the team who understands the client.

Participant 2:

"dedicated testers should be the client, should know exactly what they want and ensure that the system meets all other quality requirements before the client sees it"

This is important because it is not always feasible to have an onsite customer. The absence of dedicated testing personnel therefore results in quality requirements being looked at too late.

In essence, solid user representation, concrete guidance, and dedicated testing are the core aspects towards meeting user expectations. The absence of dedicated testing has undesirable consequences as stated in the challenges sections. The lack of concrete guidance means the team has to innovate, adopt, and devise processes, practices and techniques for ensuring the team meets user expectations. All these require that a team clearly understands the user expectations which is possible through solid user representation. These aspects require strong commitment from top management to provide an enabling environment with adequate resources and time.

CHAPTER 7. DISCUSSION

This discussion aims to integrate the concepts outlined in the findings with extant literature on method tailoring, Scrum's empirical process control, and agility. The central theme proposed in this discussion relates to the core category – "Meeting User Expectations" with a particular attention to its core concept "Lack of Concrete Guidance". Abrahamsson et al.(2003, p. 4) explain concrete guidance as "practices, activities and work products at the different phases of the software development life-cycle that characterise and provide guidance on how a specific task can be executed". This definition is extended here to include guidance on how to organise SQA activities, how to setup a Scrum project team to include SQA personnel, and how to customise the development process to incorporate SQA practices and techniques.

Since Scrum does not provide concrete guidance (Abrahamsson et al., 2003), this study proposes that Scrum needs to be seen as a framework of 'empty buckets' which need to be 'filled' with situation specific practices and processes. This study reveals that organisations need to understand that Scrum offers a planning and control backbone but needs to be supported by other methodologies. This means that organisations need to tailor their Scrum process innovatively based on the demands of the situation. Ensuring that relevant SQL practices are included in the Scrum process structure would be an innovative way of tailoring the Scrum process. A failure to understand that the Scrum package lacks details on technical SQA practices, on team composition, and work coordination can lead to various challenges.

Literature on method tailoring (Fitzgerald, Russo, & O'Kane, 2003) is integrated in this discussion to situate the method incompleteness and how this can be overcome. This literature also sheds light into how method tailoring was achieved in real world projects. In order to achieve this discussion, the chapter interweaves literature and research findings to:

- Emphasise the need for tailoring of Scrum to incorporate project specific, and situation-appropriate SQA techniques and processes.

- Discuss each of the research findings categories by drawing on method tailoring literature and the relevant extant literature specific to that category.
- Illuminate aspects of SQA in Scrum by focusing on agility and SQA from extant literature.
- Illuminate aspects of SQA in Scrum by focusing explicitly on Scrum's empirical process control.

Method tailoring literature was deemed relevant to this study because the lack of prescriptions is a common characteristic of Agile methods as their founders avoided prescribing bulky and time-demanding processes (Fitzgerald, Hartnett, & Conboy, 2006). Prescriptions provided by Agile methods are 'just enough' to add value to the software product and keep the development process as lean as possible. This study corroborates the notion that the software development community has generally accepted that no one method is comprehensive enough to provide exact fit for all types of information systems development projects (Conboy & Fitzgerald, 2007). In addition to methods being incomplete, a 1998 study by Firtzgerald (as cited in, Conboy & Fitzgerald, 2007) uncovered that rigorous use of methods in practice is limited with a reported 6 per cent of developers 'religiously' adhering to methods. It is therefore advisable to organisations who are new to Scrum or wishing to implement Scrum to be aware of the incompleteness of the method.

Further, Abrahamsson, Conboy, and Wang (2009) state that agility is contextual and situation specific. This means that organisations need to reflect on their situation and decide what agility means to them (Abrahamsson et al., 2009). Abrahamsson et al. (2009, p. 282) state that "specific needs of organisations and human nature inevitably lead to diverse interpretations and implementations of a method, which in turn lead to different, sometimes surprising, effects and consequences of use of Agile methods and associated practices". Agile teams therefore need to adopt development practices that reflect the context of the project (Hoda, Kruchten, & Noble, 2010).

The chapter is not a linear side-by-side comparison of the findings with the literature. The structure of this chapter is therefore not in-line with the structure of the findings chapter. For example, the tailoring of a Scrum process at SAIT to coordinate project work dependencies as embodied by the *Work Coordination* concept is discussed through the literature on coordination theory by (Malone & Crowston, 1994). The other categories such as *Process Structure* are covered in a detailed discussion on method tailoring because tailoring includes adopting practices and designing the process workflow. The tight relationship between adopted practices and process structure makes these two categories amenable for a combined discussion. Concepts like Constant Feedback and Solid User Representation are interweaved in Section 7.6 which discusses SQA and Agility.

Lack of concrete guidance and method tailoring are the main focus of this discussion and are a part of almost all sections in this chapter.

7.1 METHOD TAILORING

The lack of prescriptions on SQA strategies and techniques as revealed by this study implies that there is a need for Scrum to be customised and tailored to the needs of individual projects. This is in line with one stream of research in software development focusing on tailoring methods to suit the development context (Fitzgerald et al., 2006). According to Fitzgerald et al. (2006), factors that should be considered when deciding how to customise development methods include organisational issues, distributed teams, and existence of legacy systems.

Two traditional strands of research closely related to method tailoring are Method Engineering and Contingency Factor approaches (Fitzgerald et al., 2006).

"Method engineering requires a meta-method process from which precise project specific methods are constructed based on pre-defined and pre-tested method fragments" (Fitzgerald et al., 2006, p. 201)

"The contingency factor research suggests that specific features of the development context should be used to select an appropriate method from a portfolio of methods"

(Fitzgerald et al., 2006, p. 201)

There are problems with these approaches that render them unsuitable for most organisations. First, the contingency approach requires organisations to have a repertoire of methods from which to choose (Fitzgerald et al., 2006). This implies that software organisations would have to go through a learning phase of additional methods in order to be versed in each of the methods. This might not be advisable for companies that are still trying to perfect their Scrum implementation. The other problem that could make it difficult for organisations is that the experience needed to be versed in a method is best gained through development projects. Changing methods and learning a different method for every client project can be risky and costly.

Second, the method engineering approach poses problems to organisations because it requires a repository to store method fragments (Fitzgerald et al., 2006). This approach also requires the method fragments to have been tested in development projects and certified to work. Further, following this approach might force organisations to employ method engineers and may not be favourable in most organisations (Fitzgerald et al., 2006). The cost and risks of maintaining method fragments also makes method engineering not suitable to software companies.

In light of the weaknesses in method engineering and contingency factors approaches, the researcher sought literature grounded on real world case studies from organisations that have successfully implemented method tailoring as described in Fitzgerald et al. (2003) Conboy and Fitzgerald (2007) and Fitzgerald et al. (2006). The method tailoring approaches reported in these literature accounts involved establishing and using an overarching method at a macro-level for all projects and then customising the method at a micro-level to suit the needs of individual projects. This form of customising requires organisations to outline the general approach and its fundamental process elements to be followed in all projects. This strategy rises above the downfall common to both method engineering and contingency approaches which force organisations to "wait while a lengthy tailoring process takes place" (Fitzgerald et

al., 2003, p. 69). It also overcomes the need to have a series of methods and competence in each of the methods. It presents an opportunity to design one broad macro-method that is comprehensive enough to cater for majority of projects. In this way, fine-tuning the general method at a project level is easy and can be accomplished without a waste of resources and time.

Fitzgerald et al. (2006) and Fitzgerald et al. (2003) report on an approach to method tailoring which could be suitable as an alternative to the method engineering and contingency factors. Through this approach, a method such as Scrum is customised at a macro level as an over-arching approach upon which further micro-level customisation is done based on the needs of individual projects. This is partly similar to the gradual approach to tailoring revealed by this study which also included subsequent efforts at incorporating practices from other methodologies within the Scrum framework.

One way of customising a Scrum process is using technical engineering practices from eXtreme Programming (XP). "XP and Scrum were found to be very complementary with XP particularly useful for the technical development stages, whereas Scrum provided the necessary overall management process" (Fitzgerald et al., 2006, p. 201). This study reveals that some of the adopted practices such as Developer Testing are XP specific while others such as Code Reviews are general to software development.

It must be noted that there are conflicting opinions in the literature about fragmenting Agile methods because of reported synergistic relationship between practices of each method which makes it impossible to break up the method into independent practices. Beedle and Schwaber (as cited in Fitzgerald et al., 2006, p. 210) report that XP practices should be used as an integrated package in order to achieve full benefits. However, this study reveals a successful adoption of practices from XP. This is in line with Fitzgerald et al. (2006) who assert that the incompleteness of both XP and Scrum in terms of their coverage of the whole development process makes them complementary and thus suitable of combination as they address different aspects of software development.

7.2 PROCESS STRUCTURE

Structure denotes "the arrangement of, and relations between, the parts of something complex" (Strode, Huff, Hope, & Link, 2012, p. 1232). In this study, Process Structure denotes the relations between practices (i.e. Code Reviews), processes (i.e. Testing), and discrete life-cycle phases (i.e. Ready, In-Progress, Done) through which requirements are transformed to potentially shippable software. Traditional software development processes are designed to comply with assurance and measurement mechanisms whereas Agile processes reflect the need to adapt to variations in requirements, resources, and uncertainty (Nerur, Mahapatra, & Mangalaraj, 2005). Tailoring a Scrum process structure as evidenced in this study requires organisations to think about SQA practices to adopt, SQA roles and responsibilities, the workflow design, and to ensure that developer-testing phase is part of the workflow.

7.2.1 Adopted Practices

This study reveals that Scrum needs to be supported with quality control mechanisms to ensure desired outcomes. This view corroborates the explanation by Abrahamsson et al. (2003. p. 3) that "Scrum leaves open for the developers to choose the specific software development techniques, methods, and practices for the implementation process". According to Schwaber (2004) Scrum teams cannot realise the full benefits of Scrum until they improve their engineering practices to ensure that code written every day is checked in, built, and tested. A majority of such practices are prescribed in XP.

Schwaber and Beedle as cited (Abrahamsson et al., 2003) suggest the use of practices from other methodologies such as XP. While Scrum focuses on Agile project management, XP provides a collection of well-known software development techniques (Abrahamsson et al., 2003; Hashmi & Baik, 2007). Although a majority of authors on Agile claim that Agile methodologies embrace a set of best practices for SQA and control (Sfetsos & Stamelos, 2010), most of the well-known practices such as Test Driven Development are XP specific. XP techniques that can be adopted within a Scrum framework are test driven development, pair programming, refactoring, developer testing, and simple design (Fitzgerald et al., 2006). This

study provides evidence of successful adoption of code refactoring, developer testing, and simple designs.

Code Reviews

As mentioned in the Developer Testing section, every piece of work that gets addressed by a development team has to be peer reviewed before it can be tested. Peer code reviews provide assurance that the written code adheres to stated coding standards and is classified as an empirical process control mechanism in software development by Schwaber (2009). Although this practice provides the benefits stated, this study reveals disadvantages such as time-wasting arguments by senior developers over semantics which in most cases end up spoiling a positive team spirit.

Test Driven Development

Test Driven Development (TDD) came out as the most sought-after practice in this study. According to Ambler (2005, p. 36), TDD is an approach that emphasizes writing software tests before writing production code and enables programmers to "think about what new functional code should do before they write it". In addition to TDD, Agile developers develop and maintain tests and treat them as first-class artefacts. Crispin (2006) states that TDD allows developers write failing unit tests before writing code, and then write code to make the unit tests pass. "It also provides a safety net of tests that the programmers can run with each update to the code, ensuring that refactored, updated, or new code doesn't break existing functionality" (Crispin, 2006, p. 70). The use of TDD in Agile projects implies that developers assume both development and SQA aspects(Huo et al., 2004; Hashmi & Baik, 2007).

TDD lowers the amount of time that programmers take debugging software (Koch, 2005). With TDD, developers no longer treat testing as the right thing to do, but as the first thing to consider. In XP, development does not proceed at all until all tests pass, and this is normally used to gauge the developers' progress (Koch, 2005). Some authors, such as (Nerur et al., 2005) claim that TDD changes the role of traditional SQA functions in organisations. A systematic review of the literature by Sfetsos and Stamelos (2010) revealed that most empirical studies considered test-driven development as the most important practice for achieving quality. It is

important to note however that, most of studies evaluated Sfetsos and Stamelos (2010) focused on XP. This implies that the use of TDD in Scrum projects is still in early stages thus not much evidence exists to support the efficacy of adopting this practice in Scrum.

7.2.2 Workflow Design

The process workflow category is a core element which defines the process structure. It provides an outline of the phases through which backlog items are transformed into shippable software. This is where project level tailoring can be easily done with a view of incorporating SQA activities or processes. Teams need to decide the most important activities that need to be part of the workflow. Through method tailoring, a process workflow should be designed in such a way that successful completion of all the phases within the workflow includes important SQA practices and the task-board should also reflect these SQA practices. It should be noted that Scrum mandates that a team defines its own "Done" based on the demands of a project. The lack of prescriptions offers opportunities for innovating and incorporating best practices that are appropriate for unique situations. For example, an acceptance criteria could include that the study is tested and code reviewed before being marked as "Done"

7.2.3 Developer testing

This study reveals that developer testing as part of the process workflow means that developers have to test every single user story they address before the product owner can test that story. This is a critical component to ensuring quality of code and quality of functionality because the story is both code reviewed and developer tested at this phase by another developer before it can be moved to the next phase. The introduction of this phase as part of the workflow as evidenced in this study has important benefits such as early feedback and insightful test-driven design approach.

Although, there are benefits to developer testing, this study encountered situations where the quality of testing by developers was not satisfactory. This goes in line with Koch (205) who state that developers' testing perspective is limited and focused on one feature at a time. In most cases developers do not consider 'regression bugs' that may have been introduced by the new features. Suggestions to improve the effectiveness of developer testing include changing the

attitude of developers towards testing (Gill, 2005). One other technique discovered in this study is the use of 'emotional' motivation by project leaders to encourage proper developer testing.

The problems that arise from developer testing will be discussed in more detail the Dedicated Specialities section because these are tightly linked to the role of a Dedicated Tester.

7.2.4 Collective Code Ownership

Another characteristic of Agile software development that impacts quality is emphasis on collaboration and communication amongst team members (Sutharshan & Maj, 2010). This study highlights that collective and collaborative code ownership is important to meeting quality requirements. The idea is that if more than one person work on a feature and the team participates in design and planning sessions, then the quality of the software produced is better. This corroborates the finding by Maruping, Zhang, et al. (2009) who focused on 56 XP projects and discovered that collective ownership and coding standards contribute to improved technical quality.

Collaborative approach to development is made possible by co-location and face to face communication (Bhasin, 2012). This study reveals that all developers working on the same project need to within a touching distance to each other. A normal working day is characterised by continuous discussions amongst developers. Other studies such as (Huo et al., 2004) report smooth collaborative work between SQA groups and developers. This includes faster and light-weight, two-way communication between developers and SQA professionals in which a small piece of work is evaluated and feedback communicated back to developers (Huo et al., 2004). Further, business analysts and the user experience developers need to sit very close to the developers and this allows for quick exchange of ideas and clarification of requirements.

Collective ownership requires a shift in project management thinking from that of command and control to that of facilitating, directing, and coordinating (Nerur et al., 2005). The role of a traditional project manager is replaced with that of a ScrumMaster who is tasked to promote team work rather than individualistic ownership of duties. Although well supported in organisations that have dedicated SQA personnel, Williams and Cockburn (2003) state that

much needs to be done in terms of establishing clear relationships between development groups and SQA groups.

7.3 WORK CORDINATION AND DEPENDENCY RESOLUTION

Effective coordination is a fundamental factor for achieving project success (Strode et al., 2012). While Agile methods were designed to provide mechanisms for dealing with constant change, they place little importance on traditional coordination means such as upfront planning, comprehensive documentation, and stringent adherence to a pre-defined process. The researcher draws on Coordination Theory by (Malone & Crowston, 1994) who view coordination as a management of dependencies. “The key idea in Coordination Theory is that coordination is needed to address dependencies, which are the constraints on action in a situation” (Strode et al., 2012). This is particularly important for SQA efforts in software organisations because there are dependencies that need to be resolved between analysis work, user experience (UX) work, programming, and testing.

Coordination in this context is defined simply as “managing dependencies between activities” (Malone & Crowston, 1994, p. 90). This study reveals that Scrum tailoring initiatives for coordinating inter-dependent SQA activities in Scrum in order to ensure that the needs and constraints of project work do not affect the quality of the product. One key coordination mechanism customised based on the needs of the project and availability of development resources revealed in this study is the N + 1 innovation. This type of coordination mechanism addresses a *Producer / Consumer relationship* (Malone & Crowston, 1994, p. 93) whereby “one activity produces something that is used by another activity”. An example relationship in software development teams is that the UX developers design user interface screens and wireframes to be used by the developers. Further, the software analyst uses the output of analysis to create testing artefacts such as test cases. Software development teams need tp continuously improve their approach to this coordination strategy to ensure enough lead time for performing analysis work, testing work and user experience activities.

According to Malone and Crowston (1994) there are numerous kinds of dependencies that result from producer / consumer relationships as summarised in Table 7.2.

Dependency	Description	Implications for Development Teams
Prerequisite constraints	The consumer activity cannot be started before the producer activity has been completed. This kind of dependency requires effective notification mechanisms to ensure that the consumer actors are notified as soon as the producer has finished. This dependencies also require sequencing and tracking mechanisms.	Developers do not start on stories until analysis is done. Testing cannot start before development is finished. Drawing test cases requires finished analysis on stories. Drawing wireframes requires that analysis be done. The use of software applications such as GreenHopper software to visualise the transition of artefacts from the analysis stages until they work is signed off as shippable software. For example, software analysts and developers get notifications when stories are ready for test.
Transfer	"When one activity produces something that is used by another activity, the thing being produced must be transferred to the consumer activity". The use of 'just in time' delivery of producer artefacts is recommended to avoid the need for storage.	Stories and any supporting documentation are attached and stored in software applications such as GreenHopper. Wireframes and any UX artefacts are also attached as part of the stories on GreenHopper. The 'just in time' concept is not really relevant in this case, as storage is not a concern. However, timely delivery of

	The management of dependencies needs to ensure that consumer are not overwhelmed by a heavy inflow of artefacts to deal with. And the producers should also not be placed under pressure to deliver an overwhelming amount of artefacts within a short space of time.	artefacts is of utmost importance to ensure smooth transition between activities and timely delivery of Sprint artefacts. The Sprint estimation and planning sessions serve to ensure that the right amount of work is assigned to producers and consumers at the right time. Also, the N+1 innovation serves to ensure that enough time is allocated for all producer artefacts to be ready for consumption. The N + 2 also serves the purpose. It provides a buffer of some sort.
Usability	"Another, somewhat less obvious, dependency that must often be managed in a producer / consumer relationship is that whatever is produced should be usable by the activity that receives it".	The specifications and any supporting documents produced should be clear and comprehensive enough...and the screens designed should be workable. In the case of clarity of specifications, co-location also helps to clarify any misunderstandings.

Table 7.1 Dependency resolution

While the N+1 innovation works for analysis work and user experience, questions still to be answered on how dedicated testing personnel should be integrated going to be integrated into the project. This study could not reveal the exact nature of how Testers are supposed to be involved in a two-week sprint. The main question relates to the timing for testing work that is completed on the last day of Sprint. These testing bottlenecks and coordination of the testing work among testers and programmers have been noted in the literature (Talby et al., 2006).

7.4 DEDICATED TESTING

This study reveals various challenges were attributed to absence of dedicated testers in Agile software development teams. These include concerns around inadequate testing, capacity constraints, and lack of expertise. In addressing similar concerns, Ken Schwaber, the co-creator of Scrum, stated the following: “In response, I explained to them that a team is cross-functional: in situations where everyone is chipping in to build the functionality, you don’t have to be a tester to test, or a designer to design” (Schwaber, 2004, p. 104). The idea that anyone can test is dominant in Agile literature. However, the various challenges revealed by this study suggest that this premise can be misleading.

Schwaber’s (2004) statement that in a team “you don’t have to be a tester to test, or a designer to design” undermines the importance of professional expertise in these specialties in software development. The evidence presented in this study contradicts his statement as various challenges were attributed to the absence of dedicated testers. The notion of cross-functional teams defined as “teams of employees from different functional areas ”(Webber, 2002, p. 201) underpins team composition in Scrum. Cross-functional teams consist of members from diverse specialisations and thrive on effective coordination and collaboration mechanisms. In a case of software development teams, a cross-functional team would therefore consist of testers, programmers, designers, and analysts.

Koch (2005) states that although independent and professional testing is highly beneficial, Agile methods do not even mention testers. It is important to state that the statement is not entirely true because a Scrum cross-functional team should presumably include testers. The belief in some sections of Agile literature is that developer testing lessens the need for dedicated testing in Agile methods (Winter et al., 2008). This belief might have lead decision makers at SAIT to undermine the importance of dedicated testers and instead put all the trust in developers and analysts. With hindsight, the need for dedicated testers is clear and most of the challenges reported stemmed from the absence of dedicated testers. It is also important to note that the belief is dominant in XP literature because XP explicitly mentions developer testing as a practice.

Much has been written in practitioner literature about Agile testing and Koch (2005) states that efforts are being made in the Agile arena to understand how this can work. Cohn and Ford (2003) advise teams to carefully integrate testers in Agile projects to enable close collaboration between testers and programmers. It is suggested that SQA professionals must approach testing in Agile differently from testing in traditional methodologies (Talby et al., 2006). The approach recommended by this study is that one experienced Agile tester should be part of the team and should be responsible for coaching developers and analysts, managing the testing function, and overseeing the whole testing across all development teams. This approach aims at carefully integrating testers into development teams and is in line with the advice by Cohn and Ford (2003).

The next section discusses the concerns of stakeholders in relation to the adequacy of testing and subsequent levels of quality attributed to the absence of dedicated testers.

7.4.1 Consequences of the Lack of Dedicated Testing in Scrum

While developer testing is the main form of testing in Agile environments, the main problem is that most developers and customers who have to assume the responsibility of testing have never had a formal testing training (Koch, 2005). This was also noted in this study as testing expertise and the lack of testing knowledge base, testing guidelines are the main concerns with regards to testing. Koch suggests that traditional testers can be used in this situation to provide coaching and help in building the testing skills of developers. While this approach can contribute to good testing in Agile, Koch (2005) also notes that there is still a need for independent testing because the customers' and developers' testing perspectives do not encompass the bigger picture of SQA.

This study reveals that dedicated testers are needed for their ability to think differently than developers. Dedicated testers can induce client-centric thinking into the team, are able to think about quality from different dimensions, and would introduce wide-covering test strategies. This can be important in mitigating narrow testing by developers. Koch (2005) states that developers actively look for ways of making the code work, while testers look for ways to break the code. This is where dedicated testers can play a crucial role in ensuring that the product not

only achieves functionality and usability, but also meets the needs of the customer (Koch, 2005). Techniques for improving developer attitude towards testing include the use of emotional motivation which sensitises developers to the fact that they can effectively protect each others' careers by properly testing each others' code. This is partially similar to Gill's (2005) suggestion that improving developers' attitude towards testing is critical for effective developer testing.

In addition to the testing and quality issues, one of the concerns is capacity constraints. Capacity in this context refers to skills, experience, and time available to undertake work assignments. While Agile proponents as stated in (Talby et al., 2006; Koch, 2005) support shifting SQA responsibilities to developers, this can also lead to capacity demands and time pressures. The absence of dedicated testers means that business analysts and programmers have to undertake testing responsibilities. The result of this as evidenced in this study is that SQA tasks are almost always pushed aside.

The absence of a dedicated tester highlighted the need for top management to actively support a team in terms of resources. Testing is not the only aspect of SQA that requires top management support. For example, a development team needs to innovate and come up with situation-appropriate (Abrahamsson et al., 2003) and devise engineering practices (Schwaber, 2004, p. 107) to support Scrum's transparency. This requires management support for providing strategic leadership, promoting a collaborative culture, and creating an enabling environment for innovation. Management needs to give a development team the freedom to solve its own problems and devise the best ways to achieve its commitments (Schwaber, 2004, p. 108).

The next section discusses business buy-in as seen through the lived experiences of Scrum projects stakeholders.

7.5 BUSINESS BUY IN

The challenges and opportunities for innovation presented by the lack of concrete guidance imply that top management involvement and support is paramount. Management needs to

create an enabling environment and offer financial support in order for their project teams to meet quality requirements. Scrum needs to be directed from management through a set of clearly stated expectations and actions (Marchenko & Abrahamsson, 2008).This then means that Scrum needs to be communicated to stakeholders in a manner that highlights the fact that Scrum as a management framework does not prescribe any SQA processes, techniques, roles, and practices to use. Management need to be aware that teams need to innovate and devise engineering practices to support to overall Scrum framework.

Janzen and Saiedian (2005) mention economics as one of the factors affecting the adoption of TDD. This touches on two most important aspects for time-based organisations such as SAIT - time and cost to company. As evidenced in this study, a development team needs to be granted time to innovate, incorporate, and execute SQA practices. It is important to note that practices like TDD, regression testing, and code reviews do not only require research time, but can significantly affect the amount of time to complete features because of the required extensive tests that accompany production code.

Although innovation requires management support, Williams and Cockburn (2003) came across instances where management were not happy with Agile approaches, claiming that these approaches effectively give developers a leeway to hack. Cohn and Ford (2003) also report a case where some team members felt being micro-managed in Scrum because they interacted with their managers too often. Friis et al. (2011) discovered that one of the challenges that management faces in a Scrum environment is conceding that Scrum teams need to be left alone and not micro-managed. Scrum emphasizes self-managing teams (Schwaber, 2004) and this presents a challenge in traditional command and control as authority is pushed down to level of operational problems (Moe, Dingsøyr, & Dybå, 2010). Scrum teams need to have full control and authority to devise new approaches and solve their problems which means that management is supportive in providing space for innovation.

It is important to re-iterate that a development team needs full support and empowerment to devise ways of ensuring quality delivery. Without adequate support, the team faces challenges as evidenced through the consequences of the lack of testing expertise.

The next two sections focus explicitly on aspects of agility and empirical process control from the extant literature and how they apply to SAIT.

7.6 SOFTWARE QUALITY ASSURANCE AND AGILITY

McBreen (as cited in Mnkandla and Dwolatzky, 2006) defines Agile SQA as the flexibility to respond to changes in customer requirements. While traditional SQA methodologies use heavy inspection techniques, statistical mechanisms, and reporting processes (Bhasin, 2012; Feldman, 2005) this study highlights the importance of constant feedback which is a concept under the Guiding Principles category. Table 7.3 provides a brief comparison of traditional SQA, Agile SQA, and implications for Scrum teams.

Other features of Agile development that improve SQA include short-time delivery (Huo et al., 2004). This results in rapid feedback, simplicity, and constant testing (Abrahamsson et al., 2002). The design of the Process Structure as evidenced in this study should be in such a way that these aspects are possible. The principles are meant to provide guidelines for production of quality software (Mangalaraj et al., 2009). Furthermore, Agile methods have additional SQA practices that are not used in traditional approaches (Huo et al., 2004). These practices include code refactoring, continuous integration, On-site customer, and the use of system metaphor as opposed to formal architecture. For examples, development team can use applications such as Hudson continuous integration server for daily builds and Git for revision and source control management. However, the team does not have an On-site customer hence the need for solid user representation.

SQA is not only about testing and feedback, but incorporates planning and control. The next section discusses a SQA perspective of Scrum's empirical process control to situate planning and control in a Scrum process.

Traditional SQA	Agile SQA	Implications for Scrum teams
Formal Review	More Flexible, informal peer reviews	Code Reviews, Stand-up meetings, developer testing, retrospectives
QA engineers	On-site customer	on-site customer if possible, developer and analyst testing
Life-cycle phases	Frequent Integration	Use of Hudson server for frequent integration
Heavy inspection and control by QA 'police'	Emphasises collaboration and communication	Co-located team, collaboration as one of the core values
Verification and Sign-off of each life-cycle phase	Short time delivery	Two-week Sprints, major releases every two sprints.
Stringent Reporting	Rapid Feedback	Constant feedback, creation of feedback loops, open communication.
A dedicated testing phase	Constant testing	Developer Testing as part of definition of done. Analyst testing after a feature is done.
Dedicated QA and Testers	Done by developers and analysts	Done by developers and analysts. Need for dedicated testing, improvement of testing expertise through training.

Table 7.2 Traditional SQA vs. Agile SQA

7.7 EMPIRICAL PROCESS CONTROL

The most important SQA aspect about Scrum is its emphasis on empirical process control (Schwaber, 2009). Examples of empirical processes include stand up meetings, and retrospectives. As discussed in the Chapter 4, the Portal Project team holds daily Scrum meetings every day and have retrospective sessions after every Sprint. Empirical processes are characterised by transparency, inspection, and adaptation. These allow those in charge to have a "bird sight view" of the aspects of a process that affect the product quality (Schwaber, 2009). Empirical process mechanisms allow stakeholders to detect any aspects of the process that are not going well early in the process. This affords the organisation an opportunity to reorganise and adjust the work so that quality standards are not affected (Schwaber, 2009). Scrum teams should adopt Code Reviews and integrate Developer Testing in the Process Structure. It must be noted that control in Scrum does not mean controlling to create what we predict. It means controlling the "work towards the most valuable outcome possible" (Schwaber, 2009, p.1).

Scrum methodology puts in place mechanisms to mitigate risks during development (Schwaber, 1995). Controls used to mitigate risks include using Backlogs. Scrum teams team have both the Sprint Backlog and the Product Backlog. The team and management must track, manage, review, modify, and reconcile the controls at every Sprint meeting (Schwaber, 1995). In addition, the Daily Scrum meeting and the Burn-Down chart provide a way to monitor team progress and manage impediments (Friis et al., 2011). Daily Scrum meetings provide an opportunity for a team to deal with inevitable changes in user requirements (Maruping, Venkatesh, et al., 2009). This offers an opportunity for a team to re-plan accordingly based on changing requirements and available resources.

Although Scrum offers empirical process control mechanisms and agility, the methodology does not cover all aspects of SQA as discovered by this study and stipulated in Chapter 6.

7.8 IMPLICATIONS FOR PRACTICE

Agile literature accounts often state numerous SQA benefits and techniques associated with migrating to Agile. Often these techniques and benefits are related to the XP methodology. Scrum on the other hand does not prescribe any SQA techniques. From a SQA perspective, Scrum is incomplete as demonstrated in this study. It is on this premise that organisations are encourage to engage on method tailoring and customisation.

Customising Scrum in particular requires organisations to adopt industry best practices from methodologies such as XP. This study has shown how practices such as Code Reviewing can be fruitfully incorporated in Scrum process workflows. The workflow design is of particular importance in terms of ensuring that SQA practices and techniques are successfully employed in Scrum. Senior leaders should focus on continuously improving the base process and ensure that SQA practices are incorporated as soon as they gain dominance in industry.

Method tailoring has several advantages as outlined in Fitzgerald et al. (2003):

- Incorporation of new ideas as they emerge in practice
- Replacement of method components which are not working well
- Continuous improvement of the base method over time
- Early design of process artefacts, techniques, and tools that are applicable to a broad range of projects
- Establishment of a concise project level tailoring approach and criteria for consistency and rigour.

CHAPTER 8. CONCLUSION

This chapter aims to provide a summary of the findings and a reflection on whether the research objectives have been met. To achieve this aim, the chapter commences with a revisit of the research objectives. After the research objectives, the core aspects of the findings are then presented in summary. After the summary, the researcher makes recommendations for future research. The recommendations are followed by a section stating the limitations and a brief account of the contribution made by this study.

8.1 RESEARCH OBJECTIVES

The researcher embarked on an exploratory and inductive theory building case study focusing on aspects of SQA in a Scrum environment through the application of Classic Grounded Theory Methodology tenets. The study sought to understand how an organisation using Scrum achieves software quality requirements and to generate a substantive theory on Scrum and SQA. It aimed at providing an understanding of SQA processes, practices, and techniques involved, the concerns of different individuals about the processes, and how SQA might be improved. The specific research objectives were:

- To identify and illuminate aspects of SQA in a Scrum environment
- To illuminate the concerns of Scrum project stakeholders in relation to SQA in a Scrum environment.

This study viewed SQA as a broad range of activities, processes, techniques employed in Agile teams to achieve software quality requirements.

8.2 SUMMARY OF FINDINGS

The study revealed a broad range of SQA aspects related to the main concern of *Meeting User Expectations*. The *Need for Concrete Guidance* on SQA strategies, techniques and processes came up as one dominant aspect necessary for *Meeting User Expectations*. Scrum does not offer concrete guidance on technical aspects of how to achieve quality requirements. Due to

the lack of concrete guidance in Scrum, a development team has to devise *Innovations* which may include adopting practices from other methodologies. The *Innovations* may also include carefully designing the *Process Structure* to accommodate the *Adopted Practices*, to make dependency resolution smoother, and to ensure a continued improvement of the base process framework. Adopting SQA practices and designing the process structure accordingly needs to be guided by 'quality-enabling' principles such as *Collaborative Ownership* and *Continuous Improvement*.

In addition to the *Need for Concrete Guidance*, two other important aspects necessary for *Meeting User Expectations* are *Need for Solid User Representation* and *Need for Dedicated Testing.* The study revealed a number of challenges related to the absence of a dedicated tester at SAIT. The absence of dedicated testing in a Scrum environment poses challenges such as increased *capacity demands* and pressure on developers and analysts to improve their skills and expertise in both analysis/development work and testing work. The increased pressure and/or the lack of required testing expertise results in a broad range of *testing and quality issues* such as *inadequate test planning*.

It is therefore important that team composition is cognisant of the fact that there needs to be at least one member in the team representing each of the software engineering disciplines that are demanded by nature of the project. This statement suggests that a user centric application requires a strong User Experience function. Similarly, a mission critical system requires strong quality control mechanisms to be adopted within the Scrum process structure. For example, this may include acquiring skilled Agile testing personnel to oversee testing.

Further, a brief literature review reveals that Scrum already provides some level of quality assurance through empirical process control. Empirical process control makes aspects of development work visible so that appropriate action may be taken to address any issues that might hamper a team's ability to meet user expectations. Also, the fact that Scrum is an agile method means that there are some inherent aspects of SQA such as constant feedback, short time delivery, and flexibility to accommodate inevitable changes in user requirements. It is important to reiterate however that some of the widely talked about SQA practices in Agile

literature such as Test Driven Development are XP practices and need to adopted into a Scrum process.

It is therefore important that Scrum is viewed as a framework of 'empty buckets which need to be filled' with situation specific practices and processes. This means that organisations need to understand that Scrum offers a planning and control backbone but needs to be supported by other methodologies. This suggests that organisations need to tailor their Scrum processes innovatively to accommodate the demands of a situation. An innovative approach to tailoring a Scrum process is important in ensuring that relevant SQA practices are part of the process structure.

Overall, the study does present various aspects of SQA in a Scrum environment and provides an understanding of how SQA can be addressed in Scrum. The lack of concrete guidance requires innovation and/or adoption of existing quality control mechanisms. With emphasis on method tailoring the study does provide an understanding on how to improve SQA processes and techniques.

The next section presents recommendations.

8.3 FUTURE RESEARCH

This study was an exploratory account and covered a broad range of aspects. There is need for future work to focus on each of the concepts in the theoretical framework and provide a deeper understanding. For example, a future study can solely focus on Agile testing and explore how it may be best implemented in Scrum environment. Other studies can investigate the impact of the challenges discovered in these study to software quality. In addition, more case studies on how different organisations have tailored their Scrum processes can benefit both the practitioner and the academic communities. A future GTM study could extend the findings of this study through further comparative case studies and comparative cross-case analysis to modify and extend the theoretical framework.

8.4 RESEARCH CONTRIBUTION

The study contributes to the research community by providing an understanding how Scrum works in practices in relation to SQA. The study unearthed a range of concepts that are open for further exploration. The study dispels the common misunderstanding about SQA in Agile methods propagated by studies focusing solely on XP teams and then generalising their findings to the broader Agile umbrella. This study reveals that a majority of the common Agile SQA practices most talked about in the literature are XP practices. Even the studies that seek to verify - through experiments and statistical data - whether these practices work in practice, do so by investigating XP. It is therefore important for practitioners and researchers to be cognisant of the fact that Scrum does not prescribe a majority of the most popular agile SQA practices such as TDD.

This study further contributes to the research community by addressing the need to close the gap in studies focusing on the Scrum methodology. Further, the study fills the gap of investigating Agile methods beyond the adoption stage. Finally, the study also makes a contribution to the IS field by adding an inductive and substantive theory through the application of a grounded theory techniques.

8.5 LIMITATIONS

The study is subject to the following limitations: Firstly, the application of classic GTM techniques may not have been good enough given the inexperience of the researcher with this methodology. Secondly, the prior literature review means that the researcher did not completely adhere to the tenets of classic GTM. Third, the single case study might not have covered a broad range of possible variations in the field of SQA in a Scrum environment.

CHAPTER 9. 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*, 281–284.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods. *VTT Technical report*.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New Directions on Agile Methods : A Comparative Analysis. *25th International Conference on Software Engineering* (Vol. 6).

Adolph, S., Hall, W., & Kruchten, P. (2011). Using grounded theory to study the experience of software development. *Empir Software Eng*, *16*(4), 487–513.

Adolph, S., Kruchten, P., & Hall, W. (2012). The Journal of Systems and Software Reconciling perspectives : A grounded theory of how people manage the process of software development. *The Journal of Systems & Software*, *85*(6), 1269–1286.

Ågerfalk, P. J., Fitzgerald, B., & Slaughter, S. A. (2009). Introduction to the Special Issue Flexible and Distributed Information Systems Development : State of the Art and Research Challenges. *Information Systems Research*, *20*(3), 317–328.

Akif, R., & Majeed, H. (2012). Issues and Challenges in Scrum Implementation. *International Journal of Scientific & Engineering Research*, *3*(8), 1–4.

Alsultanny, Y. A., & Wohaishi, A. M. (2009). Requirements of Software Quality Assurance Model. *Second International Conference on Environmental and Computer Science* (pp. 19–23).

Ambler, S. (2005). Quality in an Agile World. *Software Quality Professional*, *7*(4), 30–34.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The Case Research Strategy in Studies of Information Systems. *Management Information Systems*, *11*(3), 369–386.

Bhasin, S. (2012). Quality Assurance in Agile: A Study towards Achieving Excellence. *IEEE Computer Society* (pp. 12–15).

Botella, P., Burgués, X., Carvallo, J. P., Franch, X., Grau, G., Marco, J., & Quer, C. (2004). ISO / IEC 9126 in practice : what do we need to know ? *Proceedings of the 1st Software Measurement European Forum (SMEF)*.

Caballero, E., Calvo-manzano, J. A., & Feliu, T. S. (2011). Introducing Scrum in a Very Small Enterprise : A Productivity and Quality Analysis. *Systems, Software and Service Process Improvement* (pp. 215–224).

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. *European Journal of Information Systems*, *18*, 332–343.

Charette, R. N. (2005). Why Software Fails. *Ieee Spectrum*, (September), 42–49.

Cho, J. (2008). Issues and Challenges of Agile Software Development with Scrum. *Issues in Information Systems*, *IX*(2), 188–195.

Cho, J., & Huff, R. A. (2011). MANAGEMENT GUIDELINES FOR SCRUM AGILE SOFTWARE DEVELOPMENT. *Issues in Information Systems*, *XII*(1), 213–223.

Cicmil, S., Hodgson, D., Lindgren, M., & Packendorff, J. (2009). Project management behind the façade. *Ephemera: Theory and Politics in Organization*, *9*(2), 78–92.

Cohn, M., & Ford, D. (2003). Introducing an Agile Process to an organisation. *Computer*, 74–78.

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

Conboy, K., & Fitzgerald, B. (2007). The views of experts on the current state of agile method tailoring. In T. McMaster, D. Wastell, E. Ferneley, & J. DeGross (Eds.), *IFIP International Federation for Information Processing* (Vol. 235, pp. 217–234). Boston: Springer.

Crispin, L. (2006). Driving Software Quality : How Test-Driven Development Impacts Software Quality. *IEEE Software*, *23*(6), 70–73.

Côté, M.-A., Suryn, W., & Georgiadou, E. (2007). In search for a widely applicable and accepted software quality model for software quality engineering. *Software Quality Journal*, *15*(4), 401–416.

Dingsøyr, T., & Dyba, T. (2008). Empirical studies of agile software development : A systematic review. *Information and Software Technology*, *50*, 833–859.

Dingsøyr, T., Dybå, T., & Abrahamsson, P. (2008). A Preliminary Roadmap for Empirical Research on Agile Software Development. *Agile 2008* (pp. 83–94). doi:10.1109/Agile.2008.50

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems and Software*, *85*, 1213–1221.

Eisenhardt, M. (1989). Building Theories from Case. *The Academy of Management Review*, *14*(4), 532–550.

Feldman, S. (2005). Quality Assurance : Much More than Testing. *ACM Queue*, *3*(1), 27 – 30.

Fernández, W. D. (2004). The grounded theory method and case study data in IS research : issues and design. In D. N. Hart & S. D. Gregor (Eds.), *Proceedings of Information Systems Foundations Workshop: Constructing and Criticising Australian National University* (pp. 43–59). Canberra Australia: ANU E-Press.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel shanno. *European Journal of Information Systems*, *15*, 200–213.

Fitzgerald, B., Russo, N. L., & O'Kane, T. (2003). Software Development Method Tailoring at Motorola. *Communications of the ACM*, *46*(4), 65–70.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*, 28–32.

Friis, D., Ostergaard, J., & Sutherland, J. (2011). Virtual Reality Meets Scrum : How a Senior Team Moved from Management to Leadership Results. *44th Hawaii International Conference on System Sciences* (pp. 1–7).

Gill, N. S. (2005). Factors Affecting Effective Software Quality Management Revisited. *ACM SIGSOFT Software Engineering Notes*, *30*(2), 1–4.

Glaser, B. (1965). The Constant Comparative Method of Qualitative Analysis. *Social Problems*, *12*(4), 436–445.

Glaser, B., & Holton, J. (2004). Remodeling Grounded Theory. *Forum: Qualitative Social Research*, *5*(2), 1–17.

Green, P. (2011). Measuring the Impact of Scrum on Product Development at Adobe Systems. *44th Hawaii International Conference on System Sciences* (pp. 1–10).

Green, P. (2012). Adobe Premiere Pro Scrum Adoption How an agile approach enabled success in a hyper-competitive landscape. *IEEE 2012 Agile Conference*.

Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly*, *30*(3), 611–642.

Hashmi, S. I., & Baik, J. (2007). Software Quality Assurance in XP and Spiral - A Comparative Study. *Fifth International Conference on Computational Science and Applications*.

Highsmith, J., & Cockburn, A. (2001). Agile Software Development : The Business of Innovation. *Computer*, *34*, 120–122.

Hirschheim, R., & Klein, H. K. (1985). Information systems epistemology: An historical perspective. In E. Mumford, R. Hirschheim, G. Fitzgerald, & T. Wood Harper (Eds.), *Research Methods in Information Systems* (pp. 1–18). North-Holland, Amsterdam.

Hoda, R., Kruchten, P., & Noble, J. (2010). Agility in Context. *Proceedings of the ACM international conference on object oriented programming systems languages and applications*, 1–16.

Hoda, R., Noble, J., & Marshall, S. (2012). Developing a grounded theory to explain the practices of self-organising Agile teams. *Empir Software Eng*, *17*(6), 609–639.

Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004). Software Quality and Agile Methods. *the 28th Annual International Computer Software and Applications Conference*, 1–6.

Janzen, D. S., & Saiedian, H. (2005). Test-Driven Development: Concepts, Taxonomy, and Future Direction. *IEEE Computer Society*, 77–84.

Kan, S. H., Basili, V. R., & Shapiro, L. N. (1994). Software quality: An overview from the TQM perspective. *IBM Systems Journal*, *33*(1), 4–19.

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

Kitchenham, B., & Pfleeger, S. L. (1996). Software Quality: The Elusive Target. *IEEE Software*, *13*(1), 12–21.

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS quarterly*, *23*(1), 67–93.

Koch, A. S. (2005). The Role of Testers in the Agile Methods. *SQP*, *7*(1), 33–40.

Leahmann, H. (2007). A DAPTING THE G ROUNDED T HEORY M ETHOD FOR I NFORMATION S YSTEMS R ESEARCH. *4th QUALIT Conference Qualitative Research in IT & IT in Qualitative Rsearch*.

Lehmann, H. (2010). Grounded Theory and Information Systems : Are We Missing the Point ? *Proceedings of the 43rd Hawaii International Conference on System Sciences*, 1–11.

Li, J., Moe, N. B., & Dybå, T. (2010a). Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*.

Li, J., Moe, N. B., & Dybå, T. (2010b). Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality. *Proceedings of the ESEM'10 Conference*, 1–10. New York: ACM.

Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, *26*(1), 87–119.

Mangalaraj, G., Mahapatra, R., & Nerur, S. (2009). Acceptance of software process innovations – the case of extreme programming. *European Journal of Information Systems*, *18*, 344–354.

Marchenko, A., & Abrahamsson, P. (2008). Scrum in a Multiproject Environment : An Ethnographically-Inspired Case Study on the Adoption Challenges. *Agile 2008* (pp. 1–12). Toronto, Canada.

Marshall, M. (1996). Sampling for qualitative research. *Family practice*, *13*(6), 522–525.

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), 377–399.

Maruping, L. M., Zhang, X., & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, *18*, 355–371.

Matavire, R., & Brown, I. (2011). Profiling grounded theory approaches in information systems research. *European Journal of Information Systems*, (35), 1–11.

Mnkandla, E., & Dwolatzky, B. (2006). Defining Agile Software Quality Assurance. *Proceedings of the International Conference on Software Engineering Advances*, 1–7.

Moe, N., & Dingsøyr, T. (2008). Scrum and Team Effectiveness : Theory and Practice. *XP 2008*, 11–20.

Moe, N., Dingsøyr, T., & Dybå, T. (2010). A teamwork model for understanding an agile team : A case study of a Scrum project. *Information and Software Technology*, *52*(5), 480–491.

Myers, M. (2009). *Qualitative research in business & management*. London: Sage Publications, Inc.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. *Communications of the ACM*, *48*(2), 72–78.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research*, *2*(1), 1–28.

Osterweil, L., Clarke, L., DeMillo, R., Feldman, S., McKeeman, B., Miller, E., & Salasin, J. (1996). Strategic Directions in Software Quality. *ACM Computing Surveys*, *28*(4), 738–750.

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*, 5452–5461.

Owens, D. M., & Khazanchi, D. (2009). Software Quality Assurance. In I. S. Reference (Ed.), *Handbook of Research on Technology Project Management, Planning, and Operations*, 245–263.

Roode, J., & Niekerk, J. C. Van. (2009). Glaserian and Straussian Grounded Theory : Similar or Completely Different ? *SAICSIT* , 96–103.

Runeson, P., & Isacsson, P. (1998). Software Quality Assurance - Concepts and Misconceptions. *Proceedings of the 24th EUROMICRO Conference, IEEE Competer Soc*, 853–859.

Schultze, U. (2000). No Title. *MIS Quarterly*, *24*(1), 3–39.

Schwaber, K. (1995). SCRUM Development Process. *OOPSLA'95 Workshop on Business Object Design and Implementation*, 1–23.

Schwaber, K. (2004). *Agile Project Management with Scrum* (pp. 1–155). Redmond, Washington: Microsoft Press.

Schwaber, K. (2009). What Is Scrum ? *Chart*. Retrieved from http://www.scrumalliance.org/resources/227

Sfetsos, P., & Stamelos, I. (2010). Empirical Studies on Quality in Agile Practices : A Systematic Literature Review. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, 44–53.

Smyth, H., & Morris, P. (2007). An epistemological evaluation of research into projects and their management: Methodological issues. *International Journal of Project Management*, *25*(4), 423–436.

Strauss, A., & Corbin, J. (1990). Grounded Theory Research: Procedures, Canons and Evaluative Criteria. *Zeitschrift fur Soziologie*, *19*(6), 418–427.

Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, *85*(6), 1222– 1238.

Sutharshan, A., & Maj, S. P. (2010). An Evaluation of Agile Software Methodology Techniques. *International Journal of Computer Science and Network Security*, *10*(12), 68–71.

Sutherland, J., Johnson, K., & Jakobsen, C. R. (2008). Scrum and CMMI Level 5 : The Magic Potion for Code Warriors. *Proceedings of the 41st Hawaii International Conference on System Sciences*, 1–9.

Sutherland, J., & Schwaber, K. (2007). *The Scrum Papers : Nuts , Bolts , and Origins of an Agile Process*, 1–181.

Talby, D., Keren, A., Hazzan, O., & Dubinsky, Y. (2006). Agile Software Testing in a Large-Scale Project. *IEEE Software*, *23*(4), 30–37.

Tan, J. (2010). Grounded theory in practice : issues and discussion for new qualitative researchers. *Journal of Documentation*, *66*(1), 93–112.

Timperi, O. P. (2004). An Overview of Quality Assurance Practices in Agile Methodologies. *T-76.650 SEMINAR IN SOFTWARE ENGINEERING*.

Urquhart, C. (2001). An Encounter with Grounded Theory : Tackling the Practical and Philosophical Issues. In E. Trauth (Ed.), *Qualitative Research in IS: Issues and Trends* (pp. 1–26). PA, USA: Idea Group Publishing, Hershey.

Urquhart, C., Lehmann, H., & Myers, M. D. (2010). Putting the "theory" back into grounded theory: guidelines for grounded theory studies in information systems. *Information Systems Journal*, *20*, 357–381.

Voss, C., Tsikriktsis, N., & Frohlich, M. (2002). Case research in operations management. *International Journal of Operations & Production Management*, *22*(2), 195–219.

Ward, W. A., & Venkataraman, B. (1999). Some Observations on Software Quality. *Proceedings 37th Annual ACM southeast regional conference*.

Webber, S. S. (2002). Leadership and trust facilitating cross-functional team success. *Journal of Management Development*, *21*(3), 201–214.

Williams, L., & Cockburn, A. (2003). Agile Software Development: It's about Feedback and Change. *Computer*, 39–42.

Winter, J., Rönkkö, K., Ahlberg, M., & Hotchkiss, J. (2008). Meeting Organisational Needs and Quality Assurance through Balancing Agile & Formal Usability Testing Results. *The 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques*.

CHAPTER 10. APPENDICES

10.1 APPENDIX A: INTERVIEW QUESTIONS

1) What is your role in relation to SQA in the Portal project?
2) How does the team ensure that Portal project meets quality requirements?
3) What highlight practices / processes either Scrum specific or in general, do you think contribute to ensuring required quality levels? What processes stand out to be the best in terms of assuring quality i.e. Coding conventions, ? And how do you think these practices can be enhanced?

3) What challenges, constraints, and problems do you think the project team faces in ensuring that required quality levels are achieved?

4) To what extent do you think quality requirements are met in the project?

5) How does the team know when quality requirements are met for a release?

6) If you were to restructure the team, how would you do it? What roles and processes would you introduce to improve quality delivery?

7) Are there any defined testing policies? Any defined testing strategy?

8) Any recommendations on how you think SQA practices must be integrated in Scrum

10.2 APPENDIX B: CONCEPT-INDICATORS (CODING SHEET)

Concept	Indicator	Data Fragment
Meeting User Expectations		
Need for Dedicated Testing	Dedicated testing has a place in scrum	I think there is place for both. I would not say there is no place for them , we have been asking for one for one for some time now,
	System Wide Testing Focus	particularly to address that issue that with regression testing, so it would be nice to not put the responsibility on the team to do a system wide test, uhm, for regression purposes, for a major release
	Testing workload sharing	I certainly agree that it does not help to have a testing team and a development team and one should hand over all the testing and they are not responsible for that , I think the way we have it now test pieces of functionality in isolation, work nicely and it is the team's responsibility to sort that out, particularly code review, so your peer reviews are very important, but functional system wide testing can be done by external dedicated testers .
	need for external testing functionality	In terms of what I would change, it would be nice to see an external testing functionality
	Help improve unit tests suit	and maybe that's the problem why I keep, why I say it might be useful to have external testing team to do these system wide tests, regression tests, possibly that's why the need for them is there is because we don't have a lot of unit tests

	Absolute need for dedicated testers	This is not an argument that you have to entertain, I think we do need dedicated testers
	Adequate expertise in testing	uhm, purely because they know what they should know when it comes to testing, you think about qualification, you are not just going to hire anyone who is going to do testing for you . and this touches on what we were saying earlier on in terms of knowing the processes...
	Specs verification and test planning	so where an analyst is running ahead, your software testers can be running in parallel with your analyst to make sure that all to specs are up to date and adhere to what to users wants
	Regression and integration testing	what you would also be testing is you always be going back in terms of regression testing, so initially in a sprint, your testers could focus on just like testing a certain requirements, not necessarily looking at stories, just about looking at the quality,
	Focus on other Quality Measures	waiting for things to be pushed into ready for test, make sure that you look at other quality measures for example perform performance testing, pressure testing, you know.
	Absolute need for dedicated Testing	If we use it as a pure debate thing, where I sit now, I believe strongly, in having SQA as a dedicated function in the environment
	Better Client Representation	Dedicated testers should be the client, should know exactly what they want and ensure that it is meeting all those quality measures before the client sees it.
	User centred thinking	I believe that kind of function could have facilitated the thinking early on in the process that didn't delay our eventual beta delay .

	Testing Repeatability and Consistency	I think u are spot on, the other part ...in testing is that it has to be consistent, and repeatable,
	Need for dedicated tester	I agree with the saying that your team is your team and they should not be divided into I do the testing and you do the coding kind of thing, but the point is, you need at least one person in that team from each of the disciplines who knows what they are doing
	Dedicated Testers provide coaching	They don't necessarily need to do it themselves, they can coach your guys and teach them how to do it because otherwise it is not gonna be done effectively
	Structured testing for integration testing	you do need some form of structured user testing just to guide your users through the process and especially if you integrating with other systems need some integration testing happening.
		testers, and analysts can operate as testers, you don't specifically need a tester, but I think it is quite naive
	System wide testing	And that is where testers are incredibly important because the testers are not just testing that the new work in itself works
	Integration and Regression Testing	but they are testing that the new work works but also works well with existing features in production
	Regression Testing Requires Dedicated Testers	another problem is regression, so bear in mind that team that I am working on does not have a dedicated tester
	user point of view in testing	In my opinion the entire story should go to a tester so that they can thoroughly test, just from an end user point of view or from a usability point of view rather than checking the quality of the code or what other code has been written,

	get a tester to test better	Even better, get them a tester. Some1 who knows how to test
Need for Concrete Guidance	Unknowingly Applying SQA	I wouldn't say we say that we know that we are applying software quality assurance . We are unknowingly applying SQA , because we know that it needs to work at the end of the day .
	Lack of Awareness of SQA techniques	I think what we are lacking then is that we are not aware of processes and techniques ,
	Lack of base guidance	I don't think we even know what we supposed to be working off as the base .
	Lack of Quality Assurance knowledge base	I think we do the best to our ability, but I don't think we understand what this whole thing means and if we doing it good
	Lack of Rules for Quality	so Scrum as a methodology does not have any rules around quality
	Lack of Prescriptions on team composition	scrum does not dictate what the team has to do within a team environment,
	Lack of Prescriptions on team composition	It just says you know, if you throw the team together, they will work it out and in my world
	Lack of guidance on how integration tests should be handled.	and i don't think scrum has any sympathy towards that there is nothing in the scrum that says well this is scrum for when you taking the product to the market and this is scrum for when you already have a product in the market
	Adopted Practices not prescribed in Scrum	But it is worth noting that Scrum says nothing about CRs. A majority of these techniques such as TDDs are associated with Scrum but are not mandated by Scrum
	Situation Appropriate strategy	You can say that at least in financial systems, if you use scrum plus this, you can expect a better quality product and so on

	Situation Appropriate Testing strategy	Although scrum offers a foundation, there is space for more work that can done. I personally would like to come across a material that gives me a testing strategy that I can readily expect to work
	Industry specific strategy	I have nt found any thing that has a significant level of support apart from individual experiences. Something like, this is what you can do with financial systems, or this is what you can do with a single scrum team
	lack of specific prescriptions	it does not give you a specific way to go about doing the work
	Analysis guidelines lacking	, particularly in the analysis side of things
	Lack of structured approach for analysis work:	But i think one thing that really hampered with our ability to get out requirements quickly was we didn't have a structured approach of how to analyse things upfront or iteratively or how that fits in the sprint
		a major lack of information, all things that I have read before about scrum say nothing about having different approach to different projects
	no testing prescriptions	"how do I use scrum to make my developers test better?" and 100% of the time, the person who answering the question who is obviously somebody using scrum will say: "Scrum cant make your developers to test better" because it doesn't prescribe how to test . If you want to find out how to test better, go and find a testing methodology and incorporate it into the phase where scrum says you need to do the testing at this point
	lack of prescribed hows	So I think that is where the problem is, without having this prescribed 'hows'….

Need for Solid User Representation	Meeting User Expectations	To your point yes, it is testing, but it goes further than that , it is not only about testing, also meeting the expectations of your user and the business owner
	User testing important	but if u didn't test your software with users then you know, it could fail as well, so therefore you don't meet the quality expectations of your user.
	Meeting user requirements	I think from where we are right now at BSG, my role from a BSD perspective is making sure that a story that you put down on the product backlog meets user's requirements ,
	Determining Requirements	you have sat down with them , you have determined the requirements, so its always like a mini spec in that story
	Accurate Articulation of Requirements	is it articulated correctly, is it clear ...
	Assessing Requirements feasibility	the second thing is, is it feasible
	getting the balance between feasibility and user needs	The role we play is facilitating that process between getting the balance between what the user wants and what is feasible
	Assessing adherence to requirements	So post that, after development and code reviews and unit testing, internal testing, when we come to do our story testing again ..." does what is been produced adhere to our story?"
	Pressure from requirements adherence assessment	The other pressure comes from, 'ok, I have asked you to do something, does it actually look like what I asked you to do?'

	Requirements-Product mismatch	I believe that is probably the biggest gap we have at the moment is that ...guys doing a bit of work here, guys are thinking they are doing the work, and when it comes to a day like this we say "ooo wait a minute, these things don't meet"
	Requirement articulation should avoid misinterpretation	So one is, of course you did not understand the requirement,
	Change Client Representation	I would change the way the client is seen, so someone like a Jo really understands what the business wants, so I would just have someone to facilitate that process and put more around deeply understands what the business wants
	Collocated user not always realistic	Ideally your team is working with your users directly so they sit with them, but that is not always realistic, you need an analyst to go do some analysis, so your stories describe the business value, but they are just showing you the what
	Design details are decided during sprinting	you still need some design aspects which you can only come through in the actual sprinting, and thats the how, and that how will obviously bring other questions that need to be answered.
	Limited user availability necessitates documentation	So yeah unless you can sit with your users 24/7 while you are coding, you going to need some form of documentation
	verifying adherence to requirements	One thing that I do think what we are not doing well, that we still need to figure out how we do it well is checking that we are doing the right thing

	poor verification processes	Thats validation, its making sure that the system actually does what it has to do . I don't think we do that very well, I think the perceived poor quality comes from there.	
	Adhering to story format gives enough insight	I think the format of the stories if adhered to allows the person writing the story to give enough insight to the developer	
	Intent and Purpose stated in story articulation	By insight we are referring to specficially to the why. What kind notable is that you can create an entire specification, 5000 pages, and never once in that document state why it is that you are building this	
	Intent and Purpose stated in story articulation	. But if you get a spec about a feature and u don't understand the intend behind that you always gonna create a , uhm, well i would say that if you know the why then u have the opportunity to add value in the value chain	
	Client interaction and availability:	the second thing was scrum it feels like, scrum relies on a lot of client of communication, or client interaction	
	Detailed articulation of a story	but behind that there is whole bunch of technical concepts, so like what are the things that i need to take into account, what are the rules and all that kind of stuff	
	User Stories not adequate	I didn't find t	user stories adequate
BUSINESS BUY-IN			

Gaining Business Buy-in	Stakeholder reluctance to grant time for TDD	I would really love to try them, but we so far down the line and there is obviously a reluctance from stakeholders to grant so much time to try to retrospectively go and write automated tests for all this legacy code although it would probably be the best way to do it. We just don't have luxury of that time
	Setting the expectations	expectation with your stakeholders and say listen, I mean you can go to our senior product owner like XXX or YYY, and say its about setting that expectation that we dont have the tools, we don't have the capability, so you are going to get what you an get what u get
	Create Awareness of the need for dedicated testing	I think what need to do is to create awareness of the need for SQA
	Superficial acknowledgement of SQA importance	everybody will say yes, we agree we do need to improve our SQA, we need to make sure it meets requirements, what we haven't gotten yet, but we getting there, is that, for long time we were not able to motivate for dedicated testers
	Enhance SQA Awareness	So I suppose the challenge is making sure that everybody is aware of why the business needs the framework. So that's a bit of a softer issue, but people need to understand exactly why QA is being brought into the framework
	Time concerns in a time-based organisation	that's where the challenge comes in...because we are time based business, that to me has almost become a driver for, we got two weeks let's get that much that in the two weeks

	Management accountable for quality outputs	I think mgt need to enable the team to perform at the level they need to perform
	Business buy in a challenge due to cost	I think it is a difficult challenge, they will say yes we all buy into the notion of having SQA, it all sounds good whatever the case is, when it comes to the cost , we need these dedicated testers its going to cost X
	Lack of business buy-in for testing resources	That's a good question, but we have managed to win that battle . I think the reason why we don't have or we didn't have a dedicated tester is that, unfortunately most people look at the short term expense versus and the long term gain is sometimes not valued
Process Structure		
Process Workflow	Use of software for visualisation	so you can configure what it takes to move an issue from beginning to done
	Testing Step in the Process Workflow	And one of the steps in our workflow is a testing process.
	Developer Testing and Code Reviews	and only once the code review is done then they will do a functional test of that work and when both the CR and FT have been done then the issue will be marked as done.
	Functional Testing	On top of that once all issues in a particular story, all activities in a story have been code reviewed and tested, then product owner does a story level test which is a higher level test of all the individual activities in a story .

		No I think we have, I think our processes have evolved, our team has been working nicely for two years, so this whole process that I described where the code reviews and functional testing are built into the work flow has worked well for us that's not something that we had when we first started
	Build testing in the workflow	But with Scrum if you build it (Testing) in, and there are various ways you can build it into your release cycles, sprint cycles, then obviously you are getting feedback sooner
	Incorporate Testing in definition of done	At the very least your definition of done should be that it is only done when it has been dev tested and tester tested
	testing as part of the workflow	In other words you should not release at the end of the sprint if testing has not been done.
Work Coordination	GreenHopper as a tool for coordination	you know how we use GreenHopper, GreenHopper allows you to set up swim-lanes
	Marking a story ready for test	So every single story that gets addressed by the team, when they finished working on it, they will mark it as ready for test,
	Upfront planning for testing	Uhm, I do believe that you need to plan for testing, as I said it may not necessarily be in a Sprint itself, it can be during the release cycle
	Test environment setup	So there is a lot of work involved in just planning testing, you need to set up test environment, you need to put dummy data
	Organising Stakeholders for testing	you need to organise your stakeholders, it calls for someone with the necessary knowledge, someone thinking with that brain to make sure that we are covering everything

	Managing end-of-sprint tasks	I think one of the difficult challenges that scrum has and I think if you look online a number of people are coming up with solutions to this, but scrum itself has not offered a solution as part of the scrum package. And that is how do you manage things that need to happen both before and after your delivery cycle?
	Analysis and development work dependencies	That itself is fine it works, but there is a problem in...say you have got 4 stories and only one analyst, now that analyst has to provide input on all four stories before the developer can start working on it
	Tester involvement	I have gone to a course, I have read the books, I followed scrum, I am scrumming but it is not working, I don't where or how my testers are supposed to get involved.
	N+1 Concept	and I will get to that essentially what i was doing is designing things upfront, preparing stories, "trying to prepare stories" like a sprint in advance is a sense, so the analysis of the story or at least the initial analysis of the story wasn't done in the cycle itself, it was done in the previous cycle
	Interaction	so in the sprint itself, the team would request an additional analysis from me or from the client to clarify things, which slowed down things because the sprint was not designed or planned to take that analysis into account

Situation Appropriate Innovations	Situation appropriate definition of Done	So you know, your software moves through a life cycle in the beginning, its all pre-release, so you may agree here that your definition of done is that it goes into the dev environment, and nothing breaks, and you do review it, and then you one 16:36 then you take it to a point where you probably
	Situation appropriate definition of Done	Then you do a revision of done to say, its only done when it is from SIT and tested for user acceptance testing, so now your testers here are not just testing in a dev environment, they are testing in the SIT environment as well
	Situation appropriate definition of Done	they don't tell you what the definition of done must be, they just say you need to agree what the definition is your environment and based on that you figure out what is acceptable
	Situation Appropriate Innovation	I appreciate that they try to cater for a broader audience as much as possible, that if you work it out you then you can be compliant but still come up with you own innovation around the core process.
	Customising scrum based on the project	when you choose how to structure your team, and when u choose how to structure your approach or project, I think it is very dependent on what the project
	Situation Appropriateness	...I think the nature of the work, and the nature of your source of requirements is coming from, define how you structure the team and how you go about things like testing cycles, test coordination, that also ties in with your release cycle and when you test stuff to make it in production

GUIDING PRINCIPLES		
Constant Ownership	Scrum ceremonies waste time	Uhm some of the challenges...Scrum I think, sometimes is a fair amount of overhead with all these scrum ceremonies, with the planning and the estimation,
	Time costly ceremonies	although it is nice for the entire team to be involved with the estimation and planning it does come at the cost,
	Scrum Ceremonies Overheads	so 1 and half days out of a two week sprint, the entire team is out of active development they are doing planning estimation and all sorts of things and this is quite a high amount of overhead that is defined by Scrum which is almost 10% of team's time
	Peer reviews necessary for meeting requirements	I a lot of the guys, I am not saying that's how it is, but the perception maybe that we need to get 36 or 40 story points done as opposed to "no" every story point that leaves my .....is of the highest quality, not because I have done it, but I have ensured that somebody else has refactored on whether I have achieved what the business wants and more
	Team Ownership of Stories	The other thing that Scrum does better is that whole uhm having not one person owning a story.... by virtue of that, people are seeing each other's code
	Code reviews for collective code ownership	you generally do have peer code reviews, so that also improves the quality because you have got more than one person doing it
	Team participation in Planning	The other thing is that your planning, your sprint planning 2, which is kinda like your design session is done with the whole team

	Combined team effort	You don't have one person running off alone figuring out how to do this thing. They might do it first before Sprint planning 2, but during the meeting they have to motivate and support their ideas and why they are doing it the way they are. So from that perspective, it does allow you to leverage from the whole team, rather than one person as a point of failure
	quality as bi product of synergy	. It keeps the team quite tight, I think u can assume that the bi-product of that is quality because if people are working better together, if there is synergy then you can expect a better output, i think .
	Code tested by other developers in the team	is that there was a round of developer testing, in other words, I tested my own code, then some1 else in the team tests my work based on the story, then the last piece of testing would the product owner testing because there is no dedicated tester
	an environment of team effort	Scrum provides the atmosphere of team effort
Constant Feedback	Early Testing	Where Scrum works well though is that because it is not waterfall, you can test stuff sooner
	Early verification and testing works well in scrum	So I mean Agile helps with that because you get stuff verified sooner, and it embraces change.
	customer gets small increments of value	the benefit of constant review cycle where the customer gets small delivered packages, of shippable software then I would say those are the main ways that scrum delivers quality

	Constant Delivery of Shippable software helps monitor budget	I would say because of the constant reviews it means that at any point in the journey, at each cycle, you have something which is shippable, it is possible to say oh well we got at 03:30... and we going to stop here thats it for now
	Agility to respond to changes to business requires	, then we can maybe either continue or at each point we can realise that our business model has changed, oh so we going to change direction, so that's one side of it
	Early Feedback	You never go so far down the road where it hasn't been reviewed and we can't turn back from that point
Continuous Improvement	Process evolution	Our process has been evolutionary, and where it has evolved for two years has worked well for us ??. I think it is good at the moment
	Process Maturity	I think our team has matured enough and the process is mature enough that we don't try to do as many as possible ,
	Estimation process maturity	we are fully aware of, you know, after two years of doing this, we have a good idea of what our velocity is, even without looking at the story points we can say that there is enough for next two weeks, I think we are pretty good at that, at our sprint targets and things
	Emphasis on continuous improvement	there should be strong emphasis within your team of continuous improvement, and the continuous improvement philosophy is that at any given stage you can always do your job better.

	Retrospectives	And that is largely driven by the retrospective meetings, that is one thing that Scrum mandates with the end goal of continuous improvement
	Retrospectives for Team Building	without continuous improvement in mind, so you could use it as a way of socialising with the team
	Retrospectives	if performed efficiently they help the team to identify areas where things could be done efficiently, or to the right level of quality and they challenge the team to then improve that.
	Retrospectives should drive continuous improvement	So without the philosophy, just having the meetings, the retrospectives in itself won't produce better quality
55	**ADOPTED PRACTICES**	
Code Reviews	Code Review before testing	and ready for test means another developer needs to first do a code review ,
	Incorporating code review feedback	if the issue is coming out of code review the issue will get send back to the original developer to sort that out
	Practice Code Reviews	and that you are doing peer code reviews,
	Value depends on team make-up	depending on the makeup of your team, code reviews become more or less important
	Improve learning	And that is how you learn u know, you have some1 who knows, looks at your code and tells you where you are going wrong. It would be reckless if we did anything other than that.

	Diminishing value of code reviews	If you have a team of 5 people senior developers, there is value in doing code reviews, but that value diminishes. So it is far less value in doing reviews than when a senior is reviewing a junior developer's code.
	Paramount for ensuring adherence to standards from juniors	So in our environment we have a lot of new grads joining so it's not even debatable, it is paramount that we have people reviewing their code, because we have people that are learning. When a senior is reviewing another senior's code, they end up just arguing over semantics. And you find that they are almost always right just that one is more correct than the other. But that can be a wasted time arguing over semantics instead of writing code.
	Less value in code reviews between seniors	So in our environment we have a lot of new grads joining so its not even debatable, it is paramount that we have people reviewing their code, because we have people that are learning. When a senior is reviewing another senior's code, they end up just arguing over semantics. And you find that they are almost always right just that one is more correct than the other. But that can be a wasted time arguing over semantics instead of writing code.
	Code reviews should pick up any errors	because the code reviews by your peers should pick up any errors you have written and your tester should find errors in the usability of your software during end-user testing
	Code Reviews and Refactoring	I don't think scrum forces you to do peer reviews. So if you choose to implement these things then you would

Test Driven Development	Fantasising about TDD	We have not done TDD, it is something that we fantasise about, particularly with our project being a legacy project
	Slim unit tests	we don't have a hell lot of tests, full stop, whether it is TDD or not, our unit tests are very slim
	Scrum needs things like TDD	So it needs to be built around things like test driven development.
	Unit Tests provide Documentation	that is a big driver for TDD and unit tests actually become your code documentation because they are very clear way of saying this is what this thing is supposed to do
	Test Driven Development	TDD definitely
	Practice Test Driven Development	that they are practicing either TDD or at least write Unit tests
Developer Testing	Developer Test Enforcement	Obviously the quality has been better since we moved to scrum , because previously we didn't have this pre-defined necessity to test every story before it makes it into production release
	Developer Testing does not have many direct codes because most of the indicators are part of the Process Workflow concept. This is because, the process workflow is designed such that developer testing is part of the workflow. Further, as indicated in code reviews, a story has to be reviewed and then developer tested. There are a few indicators in the process workflow that point to developer testing	
CHALLENGES		
Capacity Constraints	Workload Overhead	Obviously with all the testing done by the team there is an overhead.
	Required Expertise	developers are traditionally not good testers

	Simultaneously Testing and Fixing Issues	there would be enough coming out from the external testers to keep the development team busy actually fixing the issues as opposed to having to test the issues and then fix them themselves
	duality of roles	the capacity constraints , so you are asking an analyst to do an analyst job but also to be facilitating a software testers role
	pressure from duality of roles	so at one point in time, that person is going to have a dual role at the same time, which means there is going to be increased pressure on capacity required for that person
	multiple pressures	Where we trying to work in this area of multiple, uhm, I suppose pressures
	Lack of motivation to test well	I would not say it is knowledge, tools, or even skills. But I would break it down to motivation
	Lack of desire and motivation to test Well	So a coder knows that they can code and they know that they can code it well but obviously there is a desire there and motivation to code it well. A
	Attaching emotional motivation to testing	I try to highlight to the individuals of the team that when testing, it is not about finding a bug or a fault in someone else's code you can you can fix and come up with a better product or code, but what it is about is supporting and assisting your colleague. So it is more of a emotional motivation, the motivation is emotionally centred.
	Testing is a skill on its own right	I don't know if that is the std of Java developers doing the testing, because Java developers are not testers , and I personally think that testing is skill on its own right

	Lack of Role understanding	I think that analysts struggle to understand what their role is
	Regression Testing Requires Dedicated Testers	And analysts should not be the one's that are doing the testing, analysts don't specialise in doing testing in particular things like regression testing
	Jack of all, master of none	The effect is that you become the jack of all trades and a master of none
	Capacity Demands and lack of attention	you are focusing on, you are not able to do a good job because you are spreading yourselves across a view of things
	pushing testing aside	It is easy to say I will do this test case next week, or I will do this design next week you know those things have more pressing deadlines in what they require from you so it is easier to 'deprioritise' testing or limit the amount of testing that you do, it is always the first thing to do in terms of capacity
Lack of Testing Expertise	Lack of qualified testing skills	One of the challenges we looking at right now is the lack of qualified testing skills
	Lack of Knowledge Retention	I imagine if DD had to go to JHB for another project and they slotted in TT, then whatever DD had learned on the project would be lost because she developed her own SQA skills, she knows what she was doing in her little bubble , then when TT comes in, he has to develop those skills and processes from scratch .
	Lack of testing skill	I suppose the problem is we don't have that skill , I mean we don't have a proper skill for it
	Lack of testing skills and expertise	No we have a long way to go. I think the main thing is we don't have the skills and expertise

	Off-sprint testing requires testers	I think the main challenge is that we don't have the testing skills. The automated testing does help for Sprinting side of things because there you are writing code in parallel, so there it does not necessarily have to be your tester, it could be some who is part the tester part the coder, which makes them more sort of interchangeable
	Immature testing function	And also because we are so inexperienced, the idea of testing as a function in our world is a young ideaa
	Expertise can be developed	that I think can be developed, I don't think naturally an analyst would be a good tester, I think an analyst can be a good tester if he had developed the skills, potentially training or so
	Lack of experience and skills for proper testing	one is that we don't have the knowledge the skills and experience to do it
Testing Issues	Narrow testing	One of the issues that come up is that doing functional testing of these discrete activities is not necessarily doing regression testing
	Incomprehensive testing	So you will test the little piece of functionality in isolation and you could have broken something which completely unaffected by this piece of work, so that can and does happen occasionally
	testing for a predetermined outcome	because you know what you have written, and u testing for a predefined outcome, so u not necessarily testing to try to break it
	Narrow one story testing	I mean in this particular instance, MWL, we only test towards one sticky ready for test , one story ready for test,

	Inadequate Testing due to lack of understanding	lack of understanding of available tools, lack of capacity, in one sentence, you only going to test what you can test,
	Testing issues due to lack of dedicated testers	Developers do code reviews, they do some testing, so does the product owner, but I don't believe that we do thorough enough that we need something that is dedicated
	Lack of adequate Testing	I believe we had issues with quality because we hadn't properly tested each of the various scenarios based on how business will use this application.
	Lack of user orientation in testing	Because we as a team we know how to use the system, they know it in and out, they have been part of its creation, we go about testing it and using it in certain way , when u ask other people outside the organisation to use the application, to look at it, they look at it differently
	Proper assessment of adherence to requirements lacking	I don't think we always do, I think we end up focusing on the "tick all the boxes" rather than taking a step back and saying: does this thing still do what it is supposed to do in order to fulfil its function, so it kind of evolves
	users lack motivation to test	like user acceptance testing and that kind of thing, because the other problem with people is that you can tell them to play around with the system, but if they have never seen it before, and they don't know about it and they don't have any motivation, they are not going to
	Incomprehensive Testing	back there is still a bug or not comprehensive enough or you haven't done enough regression testing

	Incomprehensive Testing	again I don't think that is effective, it is not comprehensive enough
Quality Issues	Bugs slipping through	I think in most cases we meet requirements , I think there are, with every major release, one or two minor bugs that slip through as happened last night all of which have been addressed in the first hour of the day, and they all been very trivial bugs
	Product quality dependent on analysts quality assurance expertise	I mean what u put in is what you going to get out, so If I put 10% in I am going to get 10% out in terms of quality
	release delayed due quality	and secondly, and that the reason why we didn't go to our beta release , is because we were not quite happy with quality
	Approach not adequate	We went through what we believed was the approach, but the one thing that was the reason why we didn't, is because we had issues with quality
	Late attention to client quality requirements	Quality requirements are met, but they are met and looked at too late, they are almost looked at when we go to the client we say to the client and say does this meet your quality?

10.3 APPENDIX C: SAMPLE MEMO

Concept	Indicator	Data Fragment
Need for Concrete Guidance	Unknowingly Applying SQA	I wouldn't say we say that we know that we are applying software quality assurance. We are unknowingly applying SQA , because we know that it needs to work at the end of the day .

*"I don't feel OK with this being lack of expertise…the question of expertise here does not weigh for much because whether there is expertise or not…they know that it has to work at the end of the day, and somehow they get around it. This could indicate that the processes to be used during development are imprecise and not specified. In the first meeting, Head** emphasised that scrum does not prescribe anything. As such members learn the craft of making sure that the product works at the end of the day....be it learning how test, learning and experimenting with different kinds of testing frameworks, different aspects of QA processes, requirements validation, etc. Scrum says something about knowledge being gained through experiment, while this might be either good or bad, and in line with Scrum core philosophies, it might be a cause of many other issues related to SQA such as lack of testing skills, lack of guidelines and clearly defined processes. This might be due to lack of defined processes, is this inline with Scrum's empirical process control. Nothing is set on stone, so SQA in a traditional sense requires that processes and techniques be clearly defined and adhered to. Also this could also indicate that Scrum relies on individual tacit knowledge, which carries with it, the assumption that tacit knowledge can be easily shared. there are a lot of queries that can be asked around this assumption, aspects of culture, individualism, unhealthy competition, organisational elements such as bonuses and promotion, lack of trust might hinder the much required knowledge sharing that scrum relies on. Also it is not clear how the tacit knowledge is to be capture and how it is actually transferred to other team players."*

Concept	Indicator	Data Fragment
Process Workflow	Developer Testing and Code Reviews	and only once the code review is done then they will do a functional test of that work and when both the code review and functional test have been done then the issue will be marked as done.

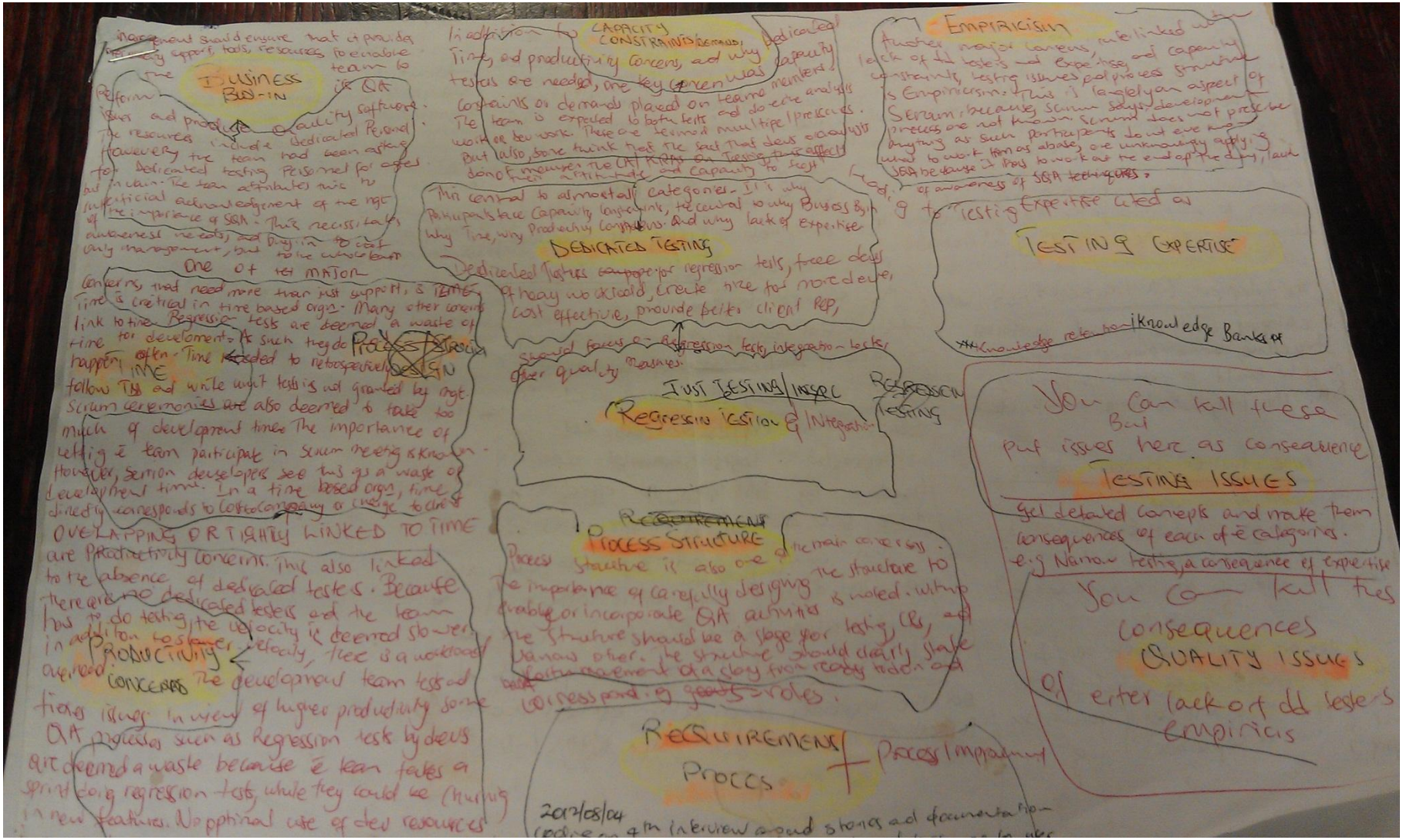
"So this can be coded in different ways, it can be about the fact that there must be a peer review on the story, and the review must ensure that development adheres to set standards. This also indicates the relationship between Adopted Practices and Process structure in that these practices are adopted and incorporated in the process workflow. So when designing the process structure, teams have to ensure that practices such as developer testing and code reviews are part of the workflow. On the other hand, this gives a definition of done, the fact that story has to have been code reviewed and functionally tested before it gets marked as done. You need to find out from developers how they feel about this, whether they think they are in a good position to do it, or whether they would rather have it done by analysts. Do they feel confident about it, do they have required business domain knowledge and supporting

documentation to perform the task, would they have it done by testers? the challenges they face on doing these two things etc".

Concept	Indicator	Data Fragment
Collective Ownership	Scrum Ceremonies Overheads	so 1 and half days out of a two week sprint, the entire team is out of active development they are doing planning estimation and all sorts of things and this is quite a high amount of overhead that is defined by Scrum which is almost 10% of team's time

"The use of development resources' time also comes into question as the respondent feel like the use of 10% of a developer time for the 'ceremonies' is just too much. This should also be seen in the context that SAIT is time based business. So cost and time again play a major role. Need to formulate relationships between collective ownership and productivity & time concerns. But this is more of a resourcing issue than productivity issue. So I need to dig deeper to explore how other respondents feel about collective ownership."

10.4 APPENDIX D: REAL TIME MEMOING AND THEORISATION



FURTHER DATA COLLECTION

- You need to decide on whether you codable a category for specific practices/processes/techniques recommended as being critical for QA! for example the importance of CRs, Unit tests, TDD, functional testing etc. Obviously, these can't stand on their own as categories. and still, Process Structure does not seem to accomodate the All. So something along the lines of Adopting QA Practices within the Workflow!

Adopted QA Practices: this as a category will also link to Process Structure, as the process workflow must be built to enable support of QA: TDD first, CRs before commits, etc.

Probe further on Documentation

→ the need or no need
→ How can you link it to Empiricism: or the central premise that scrum does not prescribe any methods to use).

→ If you are going to absorb verification into requirements Process, what part of Testing is going to be Dev if you also want to have a separate Testing

in a write up, you need to state that while Agile allows early verification, the problem is that our verification capacity is not good, we are not doing it correctly. This then leads to perceived quality issues.

One important thing

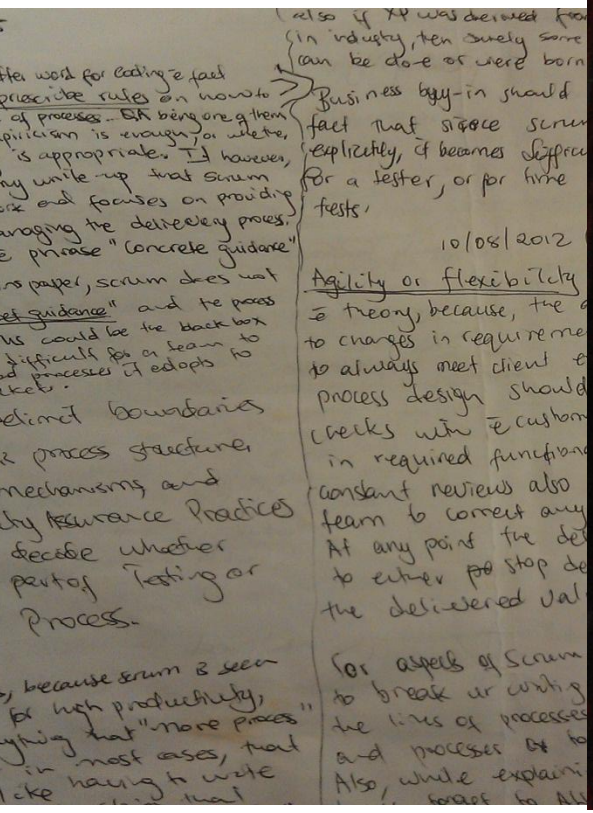
You can kill Testing as a category and introduce "Testing Coordination" This should absorb/lead the role of dedicated testers, consequences of the lack of dedicated testers, where they get involved, etc. It should also state that regression tests and integration tests should ideally be done by SD testers for effectiveness, time saving, costing perspectives etc. in this way, you can separate verification and validation that it is lacking of. This concept/category should be replaced by "Work Coordination" to clearly expand on the broader coordination of QA work: Analysis work Ntl, Tester N+0.5/-0.5 etc.

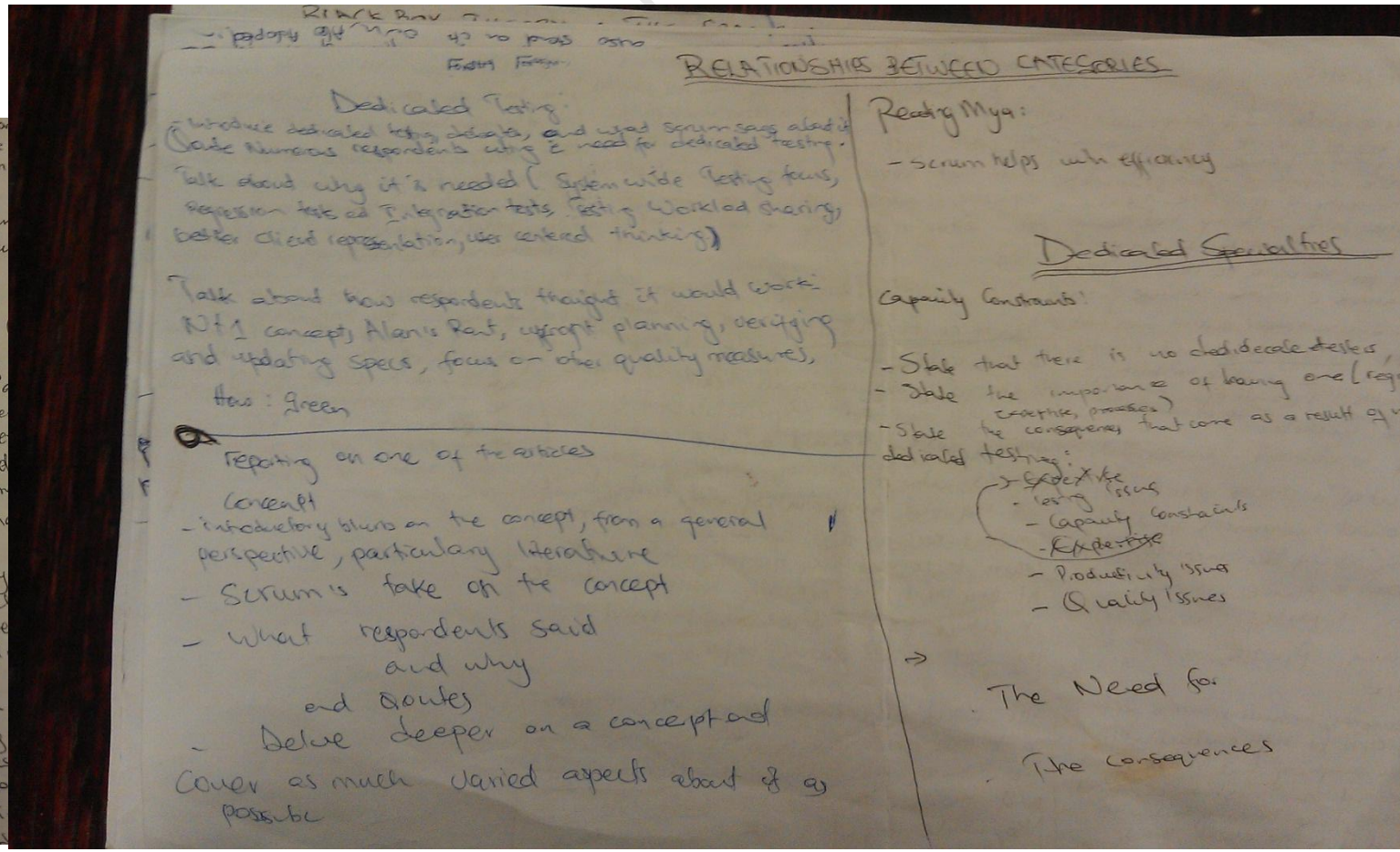
Selective Coding

-delimit the coding to only those categories that relate to the core variable in sufficient significant ways

The 6 C Paradigm Model

Context:
Conditions:
Causes:
Covariance:
Contigent:
Consequences:





RELATIONSHIPS BETWEEN CATEGORIES

Dedicated Testing:

- Introduce dedicated testing debate, and what scrum says about it
- State numerous respondents citing a need for dedicated testing.
- Talk about why it's needed (System wide Testing focus, regression tests as Integration tests, Testing Workload sharing, better client representation, user centered thinking))

- Talk about how respondents thought it would work: NOT 1 concept, Alan's Rant, upfront planning, verifying and updating specs, focus on other quality measures),

Has : Green

Reading on one of the articles

Concept

- Introductory blurb on the concept, from a general perspective, particularly literature
- Scrum's take on the concept
- What respondents said and why and doubts
- Delve deeper on a concept and

Cover as much varied aspects about it as possible

Reading Mya:

- scrum helps with efficiency

Dedicated Specialties

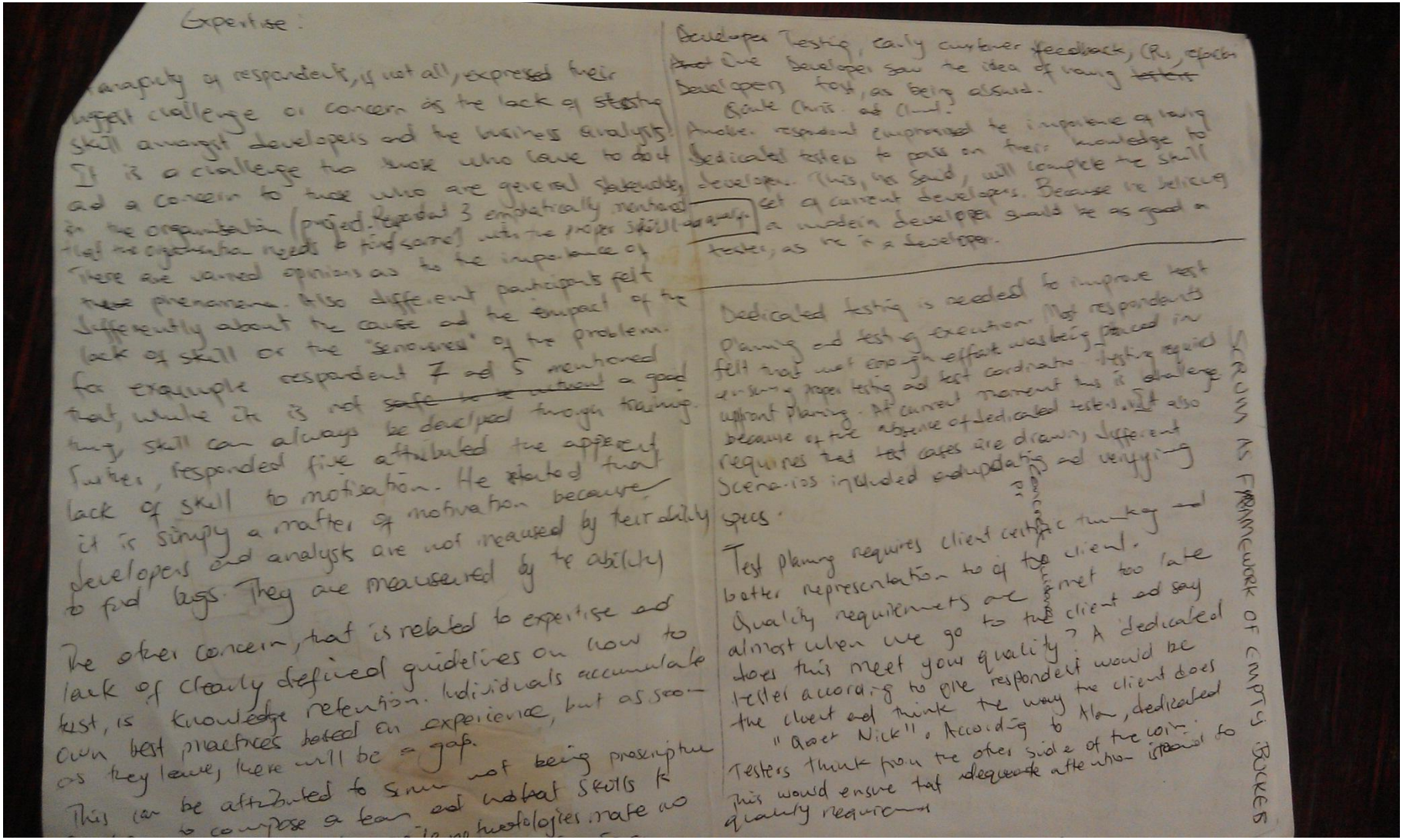
Capacity Constraints:

- State that there is no dedicated testers,
- State the importance of having one (regression tests, processes)
- State the consequences that come as a result of no dedicated testing:
  - Code testing issues
  - Capacity constraints
  - Expertise
  - Productivity issues
  - Quality issues

→

The Need for

The consequences



Expertise:

Majority of respondents, if not all, expressed their biggest challenge or concern as the lack of testing skill amongst developers and the business analysts. It is a challenge that some who have to deal with and a concern to those who are general stakeholders in the organisation (project. Regardless 3 emphatically mentioned that the organisation needs to find some) with the proper skills to adequately test. There are varied opinions as to the importance of these phenomena. Also different participants felt differently about the cause and the impact of the lack of skill or the "seriousness" of the problem. For example respondent 7 and 5 mentioned that, while it is not ~~safe to be outsourced~~ a good thing, skill can always be developed through training. Further, respondent five attributed the apparent lack of skill to motivation. He stated that it is simply a matter of motivation because developers and analysts are not measured by their ability to find bugs. They are measured by the ability

The other concern, that is related to expertise and lack of clearly defined guidelines on how to test, is knowledge retention. Individuals accumulate own best practices based on experience, but as soon as they leave, there will be a gap.

This can be attributed to some not being prescriptive to compose a team and what skills it ~~to~~ theologies make up

Developer Testing, early customer feedback, CRs, speech
~~About~~ Five Developers saw the idea of having ~~testers~~
Developers test, as being absurd.

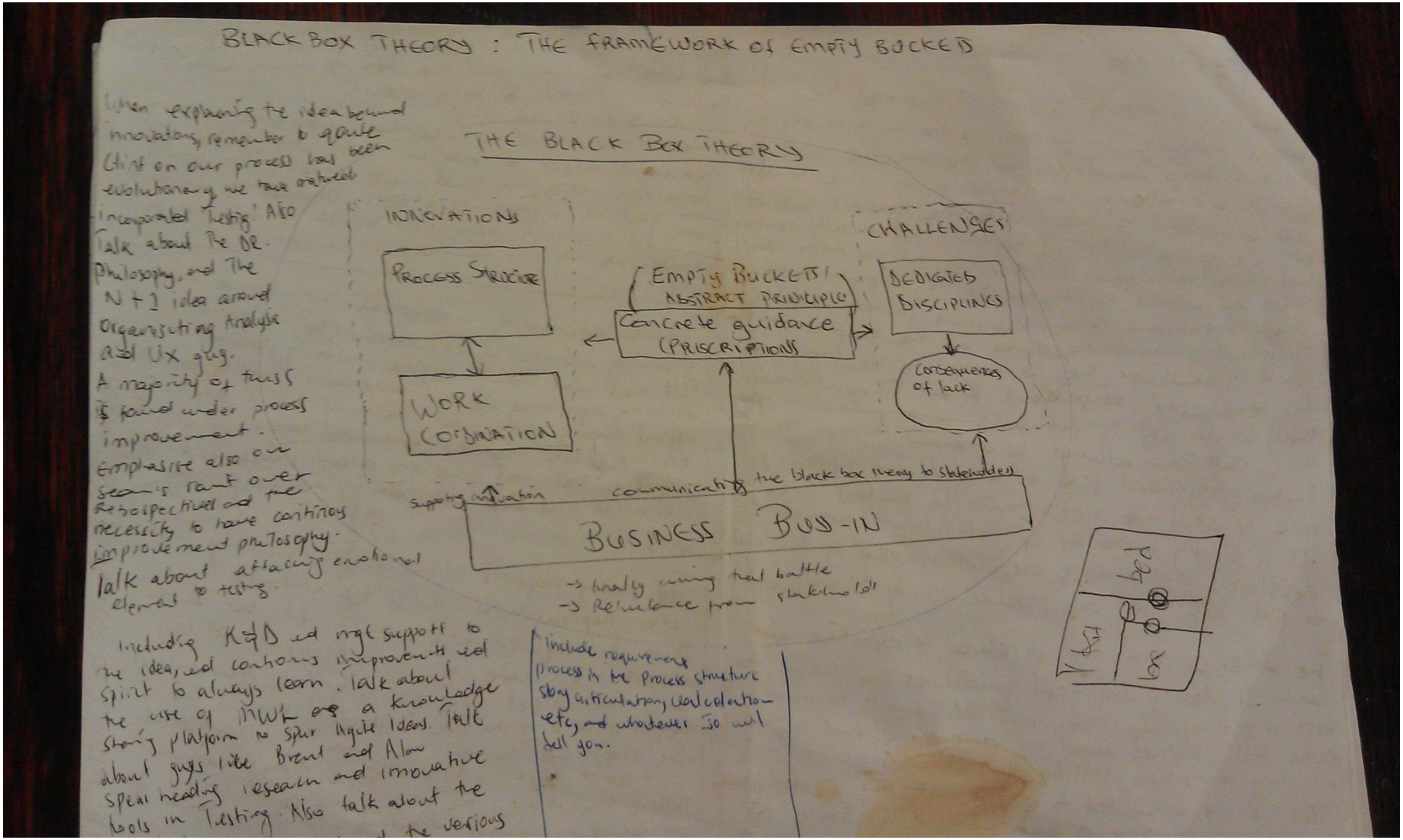
Scale Chris. as Cloud.

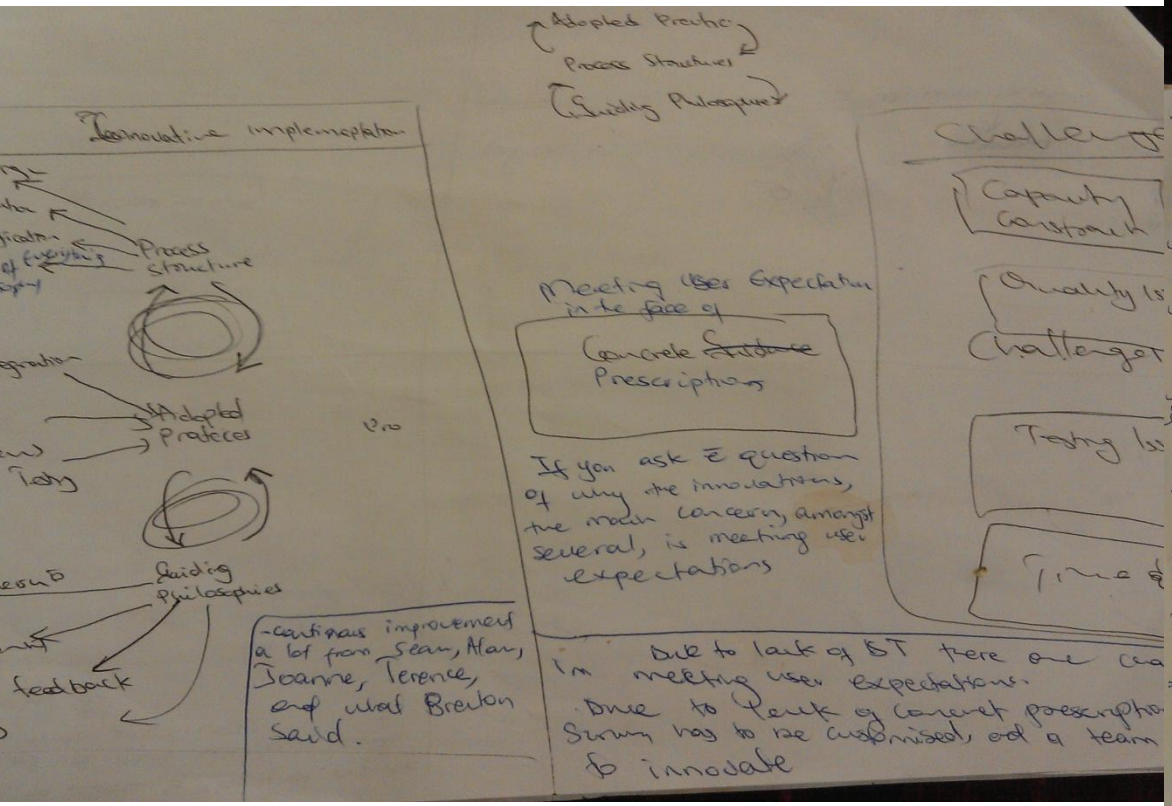
Another respondent emphasised the importance of having dedicated testers to pass on their knowledge to developers. (This, as Said, will complete the skill set of current developers. Because we believe a modern developer should be as good a tester, as he is a developer.

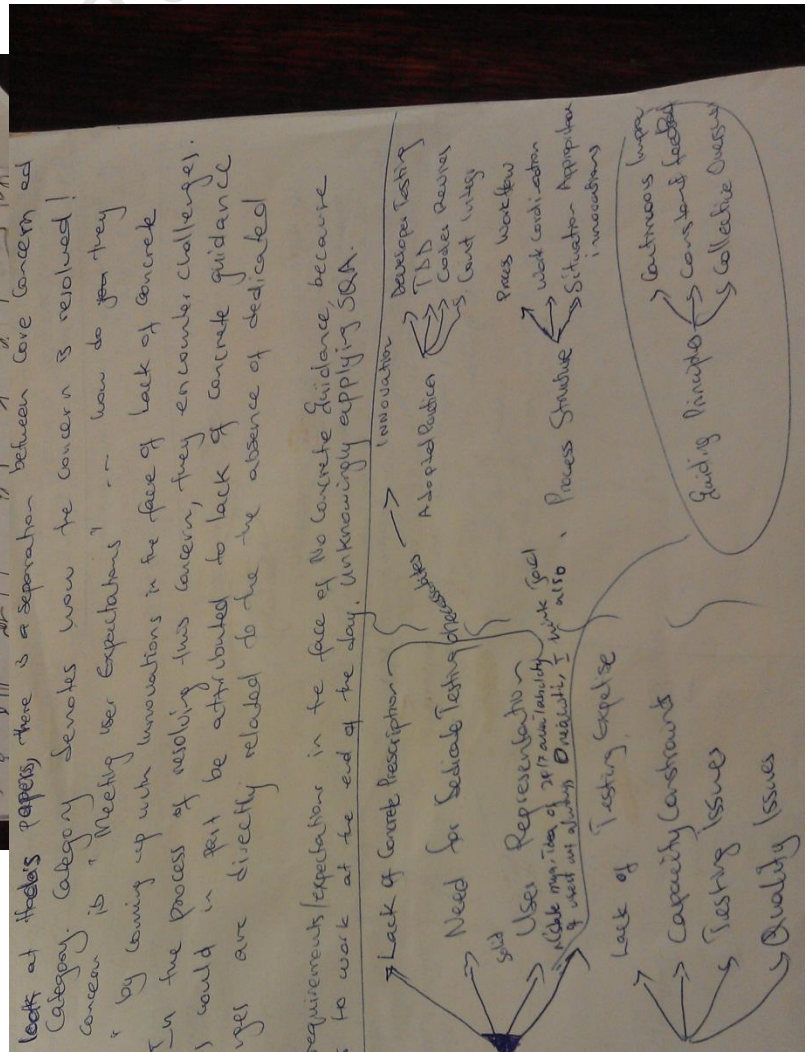
Dedicated testing is needed to improve test planning and testing execution. Most respondents felt that not enough effort was being placed in ensuring proper testing and test coordination / testing required upfront planning. At current moment this is challenge because of the absence of dedicated testers. It also requires that test cases are drawn different scenarios included exploratory and verifying specs.

Test planning requires client critique thumbs and better representation to of the client.
Quality requirements are ~~almost~~ often met too late almost when we go to the client and say does this meet your quality ? A dedicated tester according to give respondent would be the client and think the way the client does "Quest Nick". According to Alan, dedicated Testers think from the other side of the coin. This would ensure that adequate attention is given to quality requirements

SCRUM AS FRAMEWORK OF EMPTY BUCKETS


BLACK BOX THEORY : THE FRAMEWORK OF EMPTY BUCKETS
When explaining the idea behind innovations, remember to quote Clint on our process has been evolutionary we have practiced 'incorporated Testing' Also talk about the OR. Philosophy, and The N+1 idea around Organisating Analysis and Ux guy.
A majority of this is found under process improvement.
Emphasize also our team's rant over Retrospectives and the necessity to have continous improvement philosophy.
talk about attaching emotional element to testing.
THE BLACK BOX THEORY
INNOVATIONS
PROCESS STRUCTURE
WORK COORDINATION
EMPTY BUCKETS/ ABSTRACT PRINCIPLES
Concrete guidance (PRESCRIPTIONS)
CHALLENGES
DEDICATED DISCIPLINES
CONSEQUENCES of lack
supporting innovation
communicating the black box theory to stakeholders
BUSINESS BUY-IN
-> finally winning that battle
-> Reluctance from stakeholders
Including K&D and mgt support to the idea, and continous improvements and spirit to always learn. talk about the use of MVP as a knowledge sharing platform to spur Agile ideas. talk about guys like Brent and Alan Spear heading research and innovative tools in Testing. Also talk about the
Include requirement process in the process structure story articulation, real collaboration etc, and whatever Jo will tell you.


Innovative implementation
Process Structure
Adopted Practices
Guiding Philosophies
feedback
Adopted Practices
Process Structure
Guiding Philosophies
Meeting User Expectations in the face of
Concrete Prescriptions
If you ask 5 questions of why the innovations, the main concern, amongst several, is meeting user expectations
Capacity Constraints
Quality Issues
Challenges
Testing Issues
Time
-continuous improvement a lot from Sean, Alan, Joanne, Terence, and what Breton said.
Due to lack of BT there are challenges in meeting user expectations.
Due to lack of concrete prescriptions Scrum has to be customised, and a team is innovative


Lack of Concrete Prescription
Need for Scalable Testing Practices
Adopted Practices
Process Structure
Guiding Principles
Lack of Testing Expertise
Capacity Constraints
Testing Issues
Quality Issues
Collective Ownership