UNIVERSITY OF KWAZULU-NATAL

AN AGILE BASED INTEGRATED FRAMEWORK FOR SOFTWARE DEVELOPMENT

By
Sanjay Ranjeeth
972170992

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

School of Management, IT and Governance
College of Law and Management Studies

Supervisor: Professor M Maharaj

2018

DECLARATION

I , ***Sanjay Ranjeeth*** declare that

(i) The research reported in this dissertation/thesis, except where otherwise indicated, is my original research.

(ii) This dissertation/thesis has not been submitted for any degree or examination at any other university.

(iii) This dissertation/thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This dissertation/thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;
b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This dissertation/thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/thesis and in the References sections.

Signature: SRanjeeth

Date: ***08/01/2018***

ACKNOWLEDGEMENTS

I would like to express my gratitude and thanks to *Professor Manoj Maharaj* for his support and expert academic guidance in helping me to compile this thesis.

I would also like to thank my wife *Dawn* and my daughter *Lerushka* for their support and motivation and for providing an enabling environment for me to undertake this study.

Finally, I would like to express my appreciation to my mum *Shusheila* and my late dad *Robert* who were both inspirational influences in helping me to complete this journey.

LIST OF ABBREVIATIONS

ASDM	Agile Software Development Methodology
AM	Agile Methodology
BA	Business Analyst
BE	Build Engineer
CVF	Competing Values Framework
CO	Compatibility
DOI	Diffusion of Innovation
IID	Iterative and Incremental Development
IS	Information Systems
IT	Information Technology
OS	Organisational Support
PU	Perceived Usefulness
PEOU	Perceived Ease of Use
PO	Product Owner
SDM	Software Development Methodology
SAFe	Scaled Agile Framework
SE	Software Engineering
SPI	Software Process Improvement
SPM	Software Process Model
SN	Subjective Norm
SDOM	Scrum Development Operations Methodology
TAM	Technology Acceptance Model
TASDM	Theory of Acceptance of Software Development Methodology
XP	Extreme Programming

ABSTRACT

AN AGILE BASED INTEGRATED FRAMEWORK FOR SOFTWARE DEVELOPMENT

Software development practice has been guided by practitioners and academics along an evolutionary path that extends from a Waterfall approach, characterised as highly prescriptive, to an approach that is agile, embracing the dynamic context in which software is developed. Agile Methodology is informed by a set of generic principles and agile methods that are customised by practitioners to meet the requirements of the environment in which it is used. Insight into the customisation of agile methods is pivotal to uphold the evolutionary trajectory of software development methodology.

The study adopted a 'socio-technical' orientation to enhance the implementation of Agile Methodology. The social component of the study was aligned to the role played by organisational culture in the adoption of software development methodology. The amorphous concept of organisational culture has been operationalised by implementing the Competing Values Framework to develop a model that aligns organisational culture to an optimal methodology for software development. The technical component of the study has a software engineering focus. The study leveraged experiential knowledge of software development by South African software practitioners to develop a customised version of a prominent agile software development method. The model has been developed so that it is compatible with a variant of organisational culture that is aligned with agile methodology.

The study implemented a sequential research design strategy consisting of two phases. The first phase was qualitative consisting of a phenomenological approach to develop the study's main models. The second phase was quantitative, underpinned by technology acceptance theory, consisting of a survey based approach to determine South African software practitioners' acceptance of the agile-oriented technical model that was developed in the study.

The results from the survey indicated an 80% acceptance of the model proposed in study. Structural Equation Modelling was used to demonstrate that the inclusion of organisational culture as an independent construct improved the predictive capacity of technology acceptance theory in the context of software development methodology adoption. The study's overall theoretical contribution was to highlight the significance of organisational culture in the implementation of agile methodology and to extend the evolutionary path of software development methodology by proposing an agile oriented model that scales the software process to an organisational infrastructure level.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

1.1 The Challenge of Software Development

Software systems and technology in general have become pervasive to society, thereby propagating changes to societal behavior at an accelerating pace. From a pragmatic perspective, all business and social sectors of society have been transformed through the influence of software systems (Alshamrani & Bahattab, 2015; Fuggetta & Di Nitto, 2014). The societal relevance of software systems was earlier recognised by Jobs who made the prediction that software systems will not only be a major enabler in society but will also assume a defining role in people's lives (Jobs, 1995). This sentiment is echoed in Melo *et al.* (2016) and Ryan (2015) who posit that software systems are beginning to assume a pivotal role in people's personal and work oriented activities, and has a global influence on the business and economic sectors, civil and industrial infrastructures, transport, politics, education, sport and entertainment. Society has already begun to establish a 'mission critical' reliance on software systems (Fitzgerald & Stol, 2017), thereby elevating the responsibility of the software engineering community to develop software that delivers on quality and reliability on a magnitude that is befitting of the trust bestowed by society upon these systems. In this regard, the development of systems that are successful has become a necessity because it obviates the risk of economic failure as well as the risk of lives being lost. Pressman (2010) adds a new dimension by referring to the value of software systems in the production and dissemination of information on business, medical and social platforms, all of which are vital elements of the current information intensive society that we live in. Hence, there is a strong economic and social imperative to ensure that the activity of software development is refined (Fuggetta & Di Nitto, 2014) so that all possible causes of software failure are obviated enabling software systems to deliver value that is congruent with societal expectations. This can only be achieved by learning from mistakes of the past so that the activity of software development is evolved along a path that enhances the prospect of producing functional, accurate and user friendly software systems that meets the needs of society.

Historically, a de-facto methodology for software development was to follow the steps outlined in the Waterfall software process model (SPM) (Alshamrani & Bahattab, 2015). The Waterfall SPM received a lot of prominence because of its simplicity and uncomplicated, sequential nature. The simplicity of the process was based largely on an approach that involved rigid adherence to a software plan that was based on early identification of user requirements with minimal provision for adjustments to these requirements at a later stage of development. However, software development is an intricate activity reflective of the dynamics of society (Allison, 2015; Brooks, 1987; Clarke *et al.*, 2016) thereby rendering such a plan driven approach as ineffective in handling the complex requirements imposed on current software systems. In response to the shortcomings of the Waterfall SPM, the software practitioner community advocated the adoption of iterative and incremental SPM's. This transition epitomised a flexible approach that accommodated changing user requirements that were obtained iteratively rather than in accordance with a planned approach that prescribed a specific phase for the gathering of user requirements (Stoica *et al.*, 2013). The preference for an iterative approach is a deviation from a prescriptive approach enabling software systems to dynamically evolve along a path of incremental functionality. However, this change in the strategy for software development requires the invocation of a set of software development methodologies that could support such an iterative and incremental strategy. The quest for methodological support for an iterative and incremental SPM culminated in a global transition to a set of software development methods collectively referred to as the Agile Software Development Methodology (ASDM). According to Ryan (2015), this transition has been perceived as highly beneficial to the software development community and has resulted in the adoption of a software development demeanour that is agile and closely reflective of the dynamics of society thereby rendering this transition as irreversible.

ASDM is underpinned by characteristic features that embrace changing user requirements and prioritises the “visibility” (Turk *et al.*, 2014, p. 8) of a software system over comprehensive documentation and planning. The evolution of software development methodological approaches ‘trace out’ a trajectory that may be envisioned as a continuum consisting of the plan-driven Waterfall SPM on one end and the flexible, more dynamic process models that leverage ASDM attributes on the other end. These plan-driven and dynamic SPM’s have an opposing conceptual grounding, thereby presenting a challenge in respect of the transition from one SPM to the other. The extant literature on the transition from the Waterfall SPM to process models that have an agile orientation alludes to the success of the latter approach compared to the former approach (Dikert *et al.*, 2016; Dingsøyr & Moe, 2014; Ranjeeth *et al.*, 2013; Stoica *et al.*, 2013). However, there are also widespread reports of challenges that accompany an agile approach to software development. These challenges may be classified as ‘socio-technical’ in nature. The social aspect encompasses the challenge of aligning agile methodology to the prevalent organisational culture or changing the culture of an organisation so that there is an enabling organisational environment for ASDM to achieve optimal results (Fuggetta & Di Nitto, 2014). The technical challenge manifests in the ability of ASDM to scale to an organisational level or to enable organisations to customise the methodology so that it resonates with organisational processes, without compromising the principles of agility (Campanelli & Parreiras, 2015; Meso & Jain, 2006; Nerur *et al.*, 2005).

These challenges are being circumvented in an *ad hoc* manner by virtue of a ‘trial and error’ approach towards the adoption of software development methodology (Abrahamsson *et al.*, 2017), thereby compromising the value delivered by the quality of the final system. Hence, while the current evolutionary path of the software development process is enhancing the prospect of software development success, there is a need for a phenomenological inquiry into the current practice of software development. It is envisaged that such an inquiry would provide a practitioner perspective on the ‘socio-technical’ challenges of software development, thereby facilitating an informed academic contribution that guides the evolutionary path of software process improvement (SPI) techniques so that mistakes of the past are not repeated.

1.2 Background and Rationale for the Study

Historically, the activity of software development has been tagged as being embroiled in a crisis because of the inability of software developers to deliver quality software that is usable and in accordance with customer's expectations of the system (Glass, 1994; Pressman, 2010; Schach, 2008). Brook's analogous use of a werewolf to represent the problems of software development and the quest to find a 'silver bullet' that is required to slay the werewolf (Brooks, 1987) has become a seminal reference. Many software engineering experts (e.g. Boehm & Turner, 2003; Booch, 1986; Glass, 1994) use this analogy as a point of reference in their contributions to address problems related to software development. Woods (1999) as well as Griswold and Opdyke (2015) remarked that the nature of software development makes it unlikely that there will be any 'silver bullet' that will resolve the difficulties associated with developing software. This assertion is confirmed by Ingale and Jadhav (2012) and Jensen (2014) who claim that in spite of all efforts to alleviate the software crisis, software projects are still delivered late, exceed the allocated budget and are generally vulnerable to unanticipated problems. The complexities associated with software development is widely recognised by the software engineering community (e.g. Booch, 1986, 2006; Jacobson *et al.*, 1999; Pressman, 2010; Schach, 2008; Sommerville, 2007) where there is a pre-occupation with attempts to find the elusive silver bullet. However, Duggan (2004) suggests that instead of finding a silver bullet, software process improvement initiatives, referred to as 'silver pellets' could help "…tranquilize the werewolf" (p. 2) and lessen the impact of the software crisis. The idea conveyed is that the adoption of lightweight software development methodologies that are agile, adaptive and simple to implement could collectively make a defining contribution to the alleviation of the problems associated with software development.

ASDM as a less Prescriptive Alternative

Aligned to this imperative to adopt a methodology that is less prescriptive and bureaucratic, the software engineering community has advocated the implementation of ASDM as a recourse to remedy the shortcomings of the Waterfall SPM. The underlying philosophy of the agile approach is that the software development process should be less prescriptive and more reactive so that it leverages value from constant interaction with the customer with a view to efficiently accommodating changing customer requirements (Beck *et al.*, 2001). Theoretically, ASDM presents itself as methodology that is less imposing with a renewed focus on ensuring customer satisfaction as opposed to a focus on the methodological bureaucracy inherent in the Waterfall SPM. The customer-centric orientation of ASDM and the focus on speed to deliver software functionality in response to societal requirements qualify ASDM to be regarded as a silver bullet innovation in terms of software development methodology. While there has been an overwhelmingly positive response to the advent of ASDM, the attachment of silver bullet status to ASDM may be somewhat premature. There have been problems related to the implementation of ASDM in an organisational context as identified by Murphy *et al.* (2013) based on experience reports of ASDM implementation at the Microsoft Corporation.

The Success of ASDM

The growth in the popularity of agile methods is accompanied by reports of successful implementation of information systems projects that used an agile approach. The success of these interventions is confirmed by the Standish Group 2015 report that software applications developed through the agile process have approximately three times the success rate of the traditional Waterfall method (see Chaos, 2015). The acceptance of ASDM was further endorsed by the State of Agile Development survey by VersionOne (2015), that indicated a global acceptance of the agile approach as the current de-facto software process model of choice. These results are not surprising and according to Cohn (2012) the philosophy underpinning ASDM (flexibility and increased customer collaboration) could be viewed as a possible solution to the problem of failed software projects. These sentiments are similar to Kong's (2007) claim that it is the agile philosophy of embracing changing customer requirements and adopting a flexible approach to the software development process that was receiving reports of widespread success and would continue to do so. The inflexibility of traditional software development methods to handle changing user requirements in the current dynamic, information intensive society has made the agile approach a more appealing option allowing developers to leverage the flexibility inherent in the methodology. From a technical perspective, a significant shortcoming of the traditional, plan driven approaches is that as the system enters more advanced stages of development, the cost of maintenance as well as the cost (in terms of resource consumption) of changing the system requirements tends to "increase exponentially" (Schach, 2008, p. 15). However, with ASDM, the cost of changing user requirements as the system evolves over time does not have such a significant increase (Ali, 2012). This effect is illustrated in Figure 1.1.

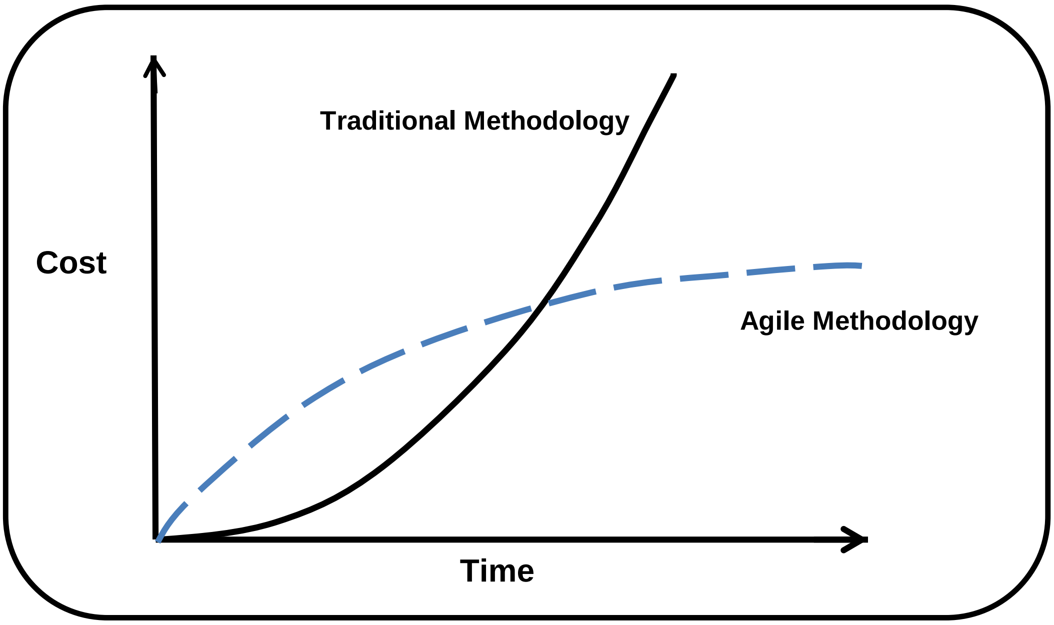


Figure 1.1: The Cost of Handling Changing User Requirements (Ali, 2012)

In spite of the reported success of ASDM, the pivotal question is whether ASDM provides the methodological 'silver bullet' that will slay the software engineering werewolf. Brooks' software engineering werewolf comprised of 4 dimensions. These include complexity (technical and management), conformity (quality standards and interface expectations), changeability (software is embedded in a mix of usability issues, laws and computing architecture that are subjected to continual change) and invisibility (software is an intangible artifact that does not have a geometric representation).

ASDM as the Silver Bullet

According to Boehm and Turner (2003), the agile approach makes a contribution in certain aspects whilst lacking in others. This view is endorsed by Ambler and Holitza (2012) who are of the opinion that ASDM may be viewed as the silver bullet for those IT projects that embrace a rapidly changing user and requirements environment. This is in keeping with the underlying philosophy of the agile approach that espouses flexibility to accommodate change and prioritises working software so that the abstractionism inherent in the software artifact is reduced from a client's perspective (Beck *et al.*, 2001). From a critical analysis perspective, these attributes of ASDM handle the changeability and invisibility dimensions of Brooks' software engineering werewolf. While this is seen as a major step in the right direction, the aspects of handling complexity and conformity associated with software development is still not catered for in the agile approach. According to Kong (2007), the less discerning implementers of the agile approach avail themselves of the flexibility inherent in the agile philosophy to adapt the methodology in an *ad hoc* manner to create customised versions of an agile methodology. The practice of technological modification resonates with the observation by Robinson and Sharp (2009, p. 211) that "…the intellectual history of an idea underpinning a technology differs from that technology's path of adoption". While this strategy is aligned to the human imperative to embrace creativity and autonomy (Fuggetta & Di Nitto, 2014) it exacerbates the problem of non-conformity with regards to software development methodology. Senapathi and Srinivasan (2012) have observed that organisations do not adhere strictly to a specific agile method, but use a tailored approach that reflects their contextual requirements from an organisational perspective.

The Need to Modify ASDM

The strategy of modifying a methodology so that it aligns to the operational environment enhances the effectiveness and usage of the methodology in the short term. Senapathi and Srinivasan warn that this pragmatic approach of adapting agile methodology has to be coupled with an understanding of the social, technical and organisational factors that influence the adoption and adaptation of agile methods in order to enhance the sustainability of the methodology. Hoda *et al.* (2010) refer to the ad hoc tampering of an agile methodology as "undercover agile" (p. 13) and they regard this activity as premature with no underlying formal framework to contextualise the existence of these ad hoc agile methods. Hence, there is a situation where organisations are adapting ASDM so that it has an 'organisational fit' that aligns the methodology to the prevalent organisational culture and technical operating environment. While this practice is theoretically aligned to the agile philosophy of flexibility and adaptability, the practical challenges of sustaining such an approach manifests in reports of scepticism of the agile approach with regards to alignment to organisational culture (Misra *et al.*, 2009; VersionOne, 2011, 2015, 2016; Wan & Wang, 2010) as well as the handling of complex software projects that need to be scaled so that it achieves the intended business value on an organisational-wide platform (VersionOne, 2011, 2016). Gualtieri (2011) asserts that ASDM is a response to software mistakes of the past and while it is perceived as a successful development methodology, the empirical evidence supporting such claims are not convincing (a claim previously made by Dybå and Dingsøyr (2008) and Erickson *et al.* (2005)). According to Gualtieri (2011), a renewed effort is required to construct a software development methodology (SDM) that embodies a modification of the agile approach so that greater coverage is attached to the 'non-coding' aspects of software development. As Boehm (2002) suggested, software engineering (SE) has a dwindling focus on computer programming and a greater focus on the economic imperative that it serves. From an ASDM perspective, Fitzgerald and Stol (2017) warn that the gains made by the transition to agility will be sub-optimal if the methodology is not adapted to satisfy the economic imperative that drives organisational behaviour.

These sentiments are a representation of the SE perspective that the agile approach is fundamentally conducive to successful software development, but it needs to be complemented with methodology guidelines or a framework that can assist in the adaptation of agile methods so that it is compatible with the organisational context. The need for methodological frameworks that inform the adaptation of ASDM is persistently sought after by the software practitioner community as is epitomised by the sentiments of SE author Robert Glass:

> *Here's a message from software practitioners to software researchers: We need your help. What help do practitioners need? We need some better advice on how and when to use methodologies. (Glass, 2004, p. 19)*

The development of frameworks that provide guidance on the use of software development methodology should however be guarded against the concomitant consequence of adding a new layer of complexity thereby negating the philosophy of simplicity and adaptability enshrined in the principles of ASDM (Beck *et al.*, 2001). A constrictive framework would also be in conflict with the assertion by Larsen *et al.* (2012) that organisations consist of a network of complex systems that are adapted by individuals in an innovative manner to enhance productivity and performance. The requirement with regards to the adaptation of ASDM is for a framework that guides the adaptation of ASDM so that it upholds the principles of agility and adds value to the broader organisational context (Dingsøyr & Lassenius, 2016). The underlying framework should take cognisance of the complexities inherent in the software development process as well as the organisational context in which software is developed. This can only be achieved by software process improvement initiatives that are implemented in an incremental manner without having a disruptive influence on the culture of an organisation.

The development of such a framework would require an understanding of the factors that influence the adoption and usage of a SDM in an organisation. According to Maciaszek (2007, p. 5) the process of software development in an organisation must be aligned with its developmental culture, social dynamics, developers' expertise, managerial practices, customers' expectations, project sizes and complexity of the application domain. From an empirical perspective, in a study involving 3200 software practitioners conducted by VersionOne (2016), organisational culture was identified as the most significant factor that influences the adoption and usage of a SDM. The influence of organisational culture on the adoption and use of a SDM has also been recognised by the academic community (e.g.Chow & Cao, 2008; Conradi *et al.*, 2000; Gallivan & Srite, 2005; Sahota, 2012; Strode *et al.*, 2009; Tolfo *et al.*, 2011).

From an overview perspective, Montoni and da Rocha (2013) provide a classification of the main factors that influence the software development process in an organisation. These were classified according to *individua*l, *organisational*, and *technological*. While the technological aspects of a software system consist of quantifiable concepts (Pressman, 2010), the influence of organisational culture and practitioner perspectives and practitioner acceptance of a particular SDM are research oriented phenomena. According to Cao *et al.* (2009), there have been instances of research efforts that have focused largely on understanding the suitability of a specific type of SDM for different project contexts (e.g.Mnkandla, 2008). While this is a step in the right direction, Cao *et al.* (2009) go on to suggest that researchers need to focus their efforts on gaining deeper insights into why the adaptation and implementation of SDM's result in either a substantial or inadequate improvement in project outcomes. According to Ryan and O'Connor (2013), one way of achieving this is by testing the underlying principles of agile methods, thereby enhancing our understanding of the human factors in the software development process. Hence, there is a need to underpin the adaptation of ASDM from a dual perspective based on knowledge from both the behavioural and technical realms of software development. This assertion resonates with Fugetta's (2000) opinion that software development is a creative, human-centered endeavor that is reliant on the precision and objectivity of an engineering methodology. According to Fugetta, software quality is reliant on the people, organisation and processes used to develop and deliver the software system. In order to engage in software process improvement initiatives, researchers need to focus their efforts on the "…complex interrelation of a number of organisational, cultural, technological and economic factors" (Fuggetta, 2000, p. 28).

Fitzgerald (1997) and Adolph *et al.* (2011) suggest that such studies involving the software process are ideally undertaken by adopting a phenomenological approach so that the real issues facing software developers are incorporated, thereby enhancing the prospect of the acceptance of the emergent view/outcome of these studies by the software development community. Such an approach would facilitate the acquisition of deeper insights into the cognitive processes that drive developer behavior. This viewpoint regarding the use of phenomenology to study software process improvement (SPI) initiatives are also endorsed by Bai *et al.* (2011) as well as (Brown, 2013).

Hence, there is a need for the academic community to develop theory that will enhance SPI initiatives by obtaining insight into the 'phenomenon' of software development as experienced by software practitioners within an organisational context. This is aligned to the call made by Basilli in his paper titled "Software Development: A Paradigm for the Future" where it is suggested that future strategy for research in software engineering is to align software engineering research with the practice of software development in a professional environment. Basili (1989) goes on to suggest that the objective of such an approach is to use the emergent view from such studies to provide an immediate enhancement to the software development process. This viewpoint is reiterated by Dingsøyr and Lassenius (2016).

The current popularity and pervasive influence of agile methods have compelled the academic community to focus SPI initiatives around ASDM with the intention of evolving the methodology based on software practitioner feedback (e.g. Brown, 2013; Lal, 2011; Montoni & da Rocha, 2013; Sidky *et al.*, 2007). In keeping with this evolutionary trajectory underpinning ASDM, the current study has the subsidiary objective of obtaining an insight into current software development practice. This insight would be pivotal in achieving the main objective of providing a framework for the adaptation of agile methods, thereby contributing to its evolution. This is aligned to the suggestion from McCormick (2001) that a viable contribution from the academic community would be to produce a meta-methodology for software development that is based on software development methodologies that have proven to be successful.

1.3 Research Questions and Main Objective of the Study

From a problem statement perspective, the main problem underpinning the current study is stated as:

How can experiential knowledge of Agile Software Development practice be used to develop a Socio-technical Framework to Guide the Implementation of Agile Software Development Methodology?

The set of sub-problems is listed below:

- What are South African software practitioners' perspectives on Agile Software Development Methodology (ASDM) from a technical perspective?
- How does organisational culture influence the implementation of ASDM?
- How can South African software practitioners’ knowledge of ASDM be used to develop a framework to guide the implementation of agile methodology?
- What is the acceptance by South African software practitioners of a framework that informs the technical implementation of ASDM?

These above-listed research questions will be used to inform the study’s main objective of developing a framework for the implementation of agile methods.

1.4 Outline of the Study

An outline of the study classified according to the chapter delineations is presented below.

Chapter 2 – Literature Review

This chapter will be guided by a philosophy that is aligned to critical hermeneutics. The evolution of software methodologies will be traced from a formalistic as well as a hermeneutics perspective. This will entail a critical inquisition of software development methodologies as it evolved from its earliest form (embodied by the 'code and fix' and the Waterfall methodology), through to its intermediate representation (prototyping and iterative and incremental models) culminating in the current state (the agile approach to software development). The objective of this chapter is to 'set the scene' for the operational elements of the current study.

Chapter 3 – The Study's Overall Design

The chapter commences with a discussion of the main 'worldview' orientations that dictate research designs in general. The discussion converges to the establishment of the researcher's philosophical assumptions and the alignment of these assumptions with an interpretivist 'worldview' orientation. The final part of this chapter is devoted to a discussion of qualitative research methodology and the relevance of using a phenomenological approach to answer the study's research questions.

Chapter 4 – The Qualitative Data

The chapter provides an insight into the qualitative data collection phase of the study. There is also a discussion of the main method of data collection, the sample used for the purposively oriented qualitative data collection phase and the criteria used for the selection of the study's sample. The chapter culminates in a discussion of the main data collection instrument and a presentation of the attributes of the study's participants from the perspectives of years of experience and the organisational sector represented.

Chapter 5 – Qualitative Data Analysis and Presentation

The main component of Chapter 5 is the qualitative data analysis. The data analysis procedures are preceded by a discussion of the techniques used to analyse phenomenological data and the implementation of these techniques in the context of the study's data. The qualitative analysis culminates in a synthesis phase referred to as the study's *trinity* that consists of a discussion of the 2 main models that are an output of the qualitative data analysis. Both the models have an alignment with the adoption of software development methodology. The first model represents a framework that links organisational culture to the adoption of a software development methodology. The second model represents an enhancement of ASDM from a technical perspective.

Chapter 6 – Quantitative Analysis

This chapter comprises of a quantitative validation of the technically oriented model derived as an output of the qualitative data analysis. The validation exercise is underpinned by technology acceptance theory and a quantitative method of data collection and analysis. The chapter culminates with a discussion of the study's data alignment to the underlying technology acceptance theoretical model. Structural Equation Modelling is used in an exploratory manner to develop a 'closer fitting' theoretical model for the study's data.

Chapter 7 – Summary and Conclusion

The final chapter will be a reflection of the achievements of the study with a focus on the study's success in answering the main research questions. There will be a presentation of the main findings of the study, the study's limitations, the contribution to the body of IS knowledge and recommendations for future research work on software development methodology.

2.0 LITERATURE REVIEW

Systematic literature reviews in all disciplines allow us to stand on the shoulders of giants and in computing, allow us to get off each other's feet (Keele, 2007, p. 12)

2.1 Introduction

One of the main reasons for engaging in a discussion of the literature is to establish what is known in the field of study thereby providing a foundation for the research topic and "…placing the study in the context of existing work" (Levy & Ellis, 2006, p. 4). Such a discussion serves the purpose of providing an insight into how the research area has become established and also provides a foundation whereby the vocabulary of the subject area may be introduced (Hart, 1998). A viable strategy for conducting a literature review is to engage in a "progressive narrowing of the topic" (Hart, 1998, p. 13) until a compelling need to solve the research problem becomes apparent. These sentiments are commensurate with the suggestion by Leedy and Ormrod (2005) that the design of a literature review should resemble an "inverted pyramid" (p. 96) also referred to as the "funnel method" (Hofstee, 2006, p. 95).

The objective of the current literature review is to provide comprehensive coverage of the main literary contributions that have underpinned academic discourse on software engineering and software development methodology. However, Hofstee (2006) observes that a comprehensive or exhaustive literature review is difficult to achieve. A possible strategy is to commence the literature review with a broad review of the major concepts and classic theories that have defined the problem domain. Cooper (1988) provides a taxonomy for the structure of a literature review that hinges on the 2 pivotal components of coverage and organisation. From a coverage perspective, a review should cover defining academic contributions that have provided direction for the domain of the study. One strategy to achieve this is by organising the review in a historical, chronological order.

The literature review for the current study will be preceded by an explanation of 2 crucial pieces of software engineering nomenclature that will be used throughout the study. The first is a reference to the software process, a set of tasks performed by software practitioners to develop quality software that meets the requirements of those who have commissioned its development (Pressman, 2010). The second is a reference to software process improvement (SPI), an expression that alludes to activities that enhance the process of software development to enable the production of higher quality software.

Architectural Design of the Literature Review

The 'architectural' design of the literature review for the current study will be arranged chronologically and it will assume an evolutionary path that is guided by SPI initiatives, as illustrated in Figure 2.1. This strategy is commensurate with the assertion by Marakas (2006, p. 4) that "…an ideal way to understand the current state of a technology is to become familiar with its evolution." The SPI initiatives have been driven by the quest to produce quality software and have followed an evolutionary path that has culminated in a philosophy and a set of software development methods collectively referred to as the Agile Software Development Methodology (ASDM). After having experienced more than a decade of agile methods, the software practitioner community is making an initiative to adapt ASDM so that a 'best practice' framework for ASDM can be developed. These initiatives are being labelled as 'Agile 2.0', a reference to the extension/adaptation of ASDM on the basis of feedback from the software practitioner community. The scope of coverage is quite expansive and provides coverage of the main software process models that have had a defining influence on the academic discourse on SPIs. This strategy is adopted so as to compensate for the lack of a solid theoretical foundation in the IS discipline (Levy & Ellis, 2006).

A sequential overview version of the literature review is presented in Figure 2.1.

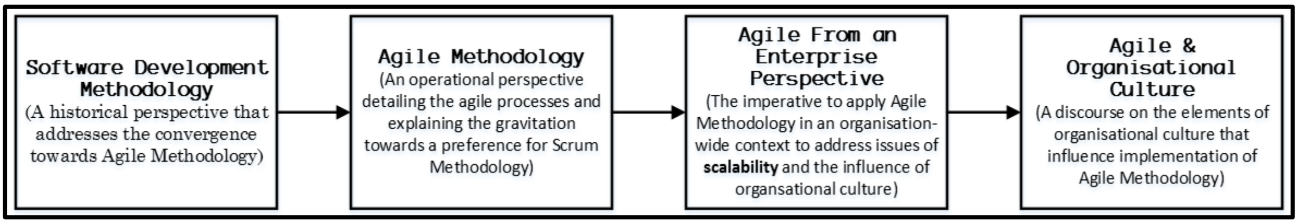

Software Development Methodology
(A historical perspective that addresses the convergence towards Agile Methodology)
Agile Methodology
(An operational perspective detailing the agile processes and explaining the gravitation towards a preference for Scrum Methodology)
Agile From an Enterprise Perspective
(The imperative to apply Agile Methodology in an organisation-wide context to address issues of **scalability** and the influence of organsational culture)
Agile & Organisational Culture
(A discourse on the elements of organisational culture that influence implementation of Agile Methodology)

Figure 2.1: Sequential Overview illustration of the Literature Review

The initial phases of the literature review have a historical slant and traces the evolution of software process models from the 'code and fix' methodology through to the iterative and incremental methodology culminating in agile software development methodology (ASDM). The coverage of ASDM is in reference to the main agile process models that have been purposively selected by virtue of usage trends. The potential for agile methodology to scale to an organisational level so that it delivers on the expected business value is currently receiving a lot of attention in the academic and practitioner sectors. This trend warrants a review of enterprise/organisational wide software process models that have an alignment with ASDM. The literature review will culminate with a discourse on organisation culture because of the influence that organisational culture has on the adoption of a software development methodology. The preceding narrative on the detail of the literature review is presented graphically as a hierarchical illustration in Figure 2.2.

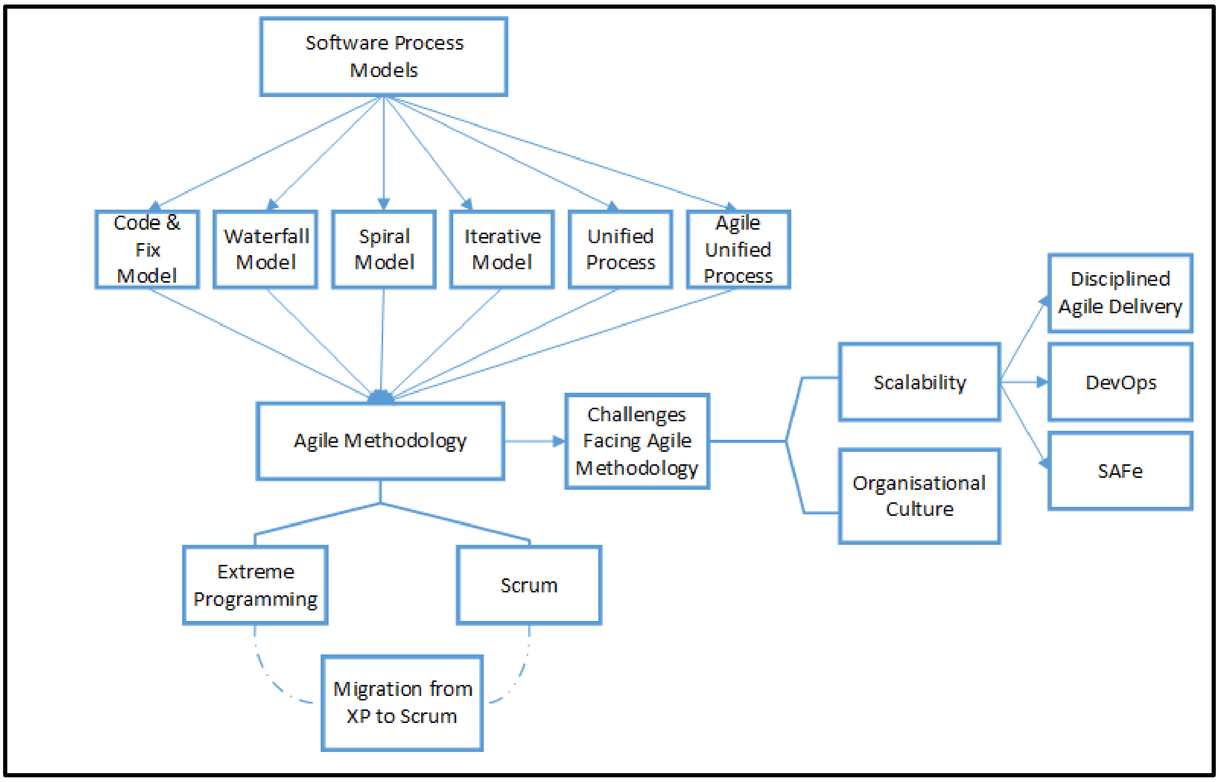


Figure 2.2: Architectural Hierarchical Design of the Literature Review

The expansive coverage of the literature on software development methodology (broadly illustrated in figures 2.1 and 2.2) provides the researcher with an opportunity to leverage this insight to find ‘gaps in the body of knowledge’.

2.2 The Software Process

According to Sommerville (1996), the software process consists of the acquisition of information about the requirements of a software system and the implementation of activities that will contribute to the development of that system. Schach (2008) formalises this interpretation by suggesting that the software process incorporates a software development methodology with an underlying software process model (SPM). Pressman (2010) provided a bit of clarity on the concept of a SPM by suggesting that a SPM is a series of predictable or repeatable steps that has to be followed in order to produce quality software. Whilst these interpretations of a SPM are adequate, one has to go back to Boehm’s elaboration of a SPM in order to fully understand its relevance to the software process. As a precursor to the introduction of the Spiral model of software development, Boehm (1988) suggested that a SPM delineates the different stages of software development and establishes different criteria that would indicate a transition from one stage to the next. Hence, it can be established that a SPM is an abstraction of the software process with clearly defined phases that a software system undergoes from its inception to completion. Transition between the phases occurs in a controlled manner. From the preceding discussion, the notion that a SPM embodies a listing of software development processes as well as a protocol regarding the sequence and transition between the phases makes the use of the term appropriate for the requirements of the current study.

The SDLC and the Waterfall SPM

The second issue that needs clarification is the interchangeable use of the SDLC model and the Waterfall SPM. The first instance of a comprehensive SDLC model was suggested by Sir Winston Royce by virtue of the Waterfall model[1] for software development proposed in Royce (1970). The Waterfall model served as a catalyst for the software engineering community to suggest adaptations and alternatives to the model based on practitioners' perspectives on the use of the Waterfall model. The dominance of this model in the 1970's and early 1980's resulted in the Waterfall model becoming a *de facto* replacement for the generic concept of a SDLC model (inferred from Aveson & Fitzgerald, 2006; Marakas, 2006; Scacchi, 1987). The Waterfall SPM became a synonym for the SDLC model. This situation was not ideal as it became an implicit impediment to the development of SDLC models that were not aligned to the Waterfall model (McCracken & Jackson, 1982; Victor, 2003). In a paper titled 'A Spiral Model for Software Development and Enhancement'[2], Boehm attempted to rectify this untenable situation. In this article, Boehm (1988) tries to re-establish the concept of a SPM (somehow lost with the collapsing of the SDLC and the Waterfall model into one and the same thing). Boehm uses the terminology SPM as a generic reference to 'cover' specific instances of SPM's such as the Waterfall, Spiral, and Iterative and Incremental models. The usage of the term SPM in the current study is strongly aligned to Boehm's interpretation. The term software process model (SPM) will be broadly used in the current study to refer to the architectural design of the software process as well as the criteria that controls traversal through the different phases of the software process. In essence, a SPM alludes to the various systems development phases found in the SDLC as well as a protocol that controls the transition between the different phases of the SDLC.

The software process is composed of a mix of a SPM and a software development methodology (Schach, 2008). Whilst the concept of a SPM has been given prominence in the preceding discussion, the concept of a software

[1] The Waterfall SPM is discussed in Section 2.3.2

[2] The Spiral SPM introduced by Barry Boehm is discussed in Section 2.3.4

development methodology (SDM) needs to be clarified. George *et al.* (2004, p. 24) regard a SDM as a "step-by-step description" of the process of developing an information system or as Schach (2008) describes it simply as the strategy used to develop a software system. However, once more it was Boehm who provided some clarity by contextualising a SDM with reference to a SPM. Boehm (1988) suggested that a SDM provides guidance on how to 'navigate' through any specific phase of a SPM. This navigation may be facilitated by software development techniques such as stepwise refinement, flowcharting, structured analysis and design (commonly referred to as the classical paradigm of software development) and object-oriented analysis and design. For the purpose of the current study, the use of the terms software process model (SPM), software development methodology (SDM) and software development technique is aligned to Boehm's interpretation of these concepts as presented in the preceding discussion. The preceding paragraph serves the purpose of:

- introducing the concept of a SPM and how it is interpreted as part of the current study;
- providing clarity on the distinction between a SDLC model, a SPM and the Waterfall model for software development;
- providing an introduction to the concept of a SDM and its relevance to the software process.

The concepts introduced in the preceding paragraph play a pivotal role in the software process. A discussion of these concepts was required in order to facilitate the presentation of a cogent discussion on the evolution of SPM's (conducted in the next section).

2.3 Software Process Models and Paradigms of Software Development – A Historical Perspective

Lonchamp (1993) stresses on the importance of introducing the nomenclature used for deliberations regarding SPM's in a pragmatic way so that it enables a precise discourse on SPI initiatives. The strategy adopted in the current study is to provide a critical overview of the traditional SPM's thereby establishing a foundation from which SPI initiatives with regard to ASDM may be explored with minimal distraction from a terminology perspective. This strategy also provides a firm foundation from which the evolutionary trajectory of SPM's may be understood, thereby enabling a deeper appreciation of the nuances that provide a distinction between different SPM's.

2.3.1 The Code and Fix Model and the Software Crisis

One of the earliest software process models (late 1940s to early 1960s) is referred to as the 'code and fix' method (Boehm, 1988) that entailed a simple 2-step process (Figure 2.3) of writing some code and fixing errors that may be observed when the program is run. This process is repeated until the software solution produces an accurate output that conforms to the expected output from the test case values that are input into the software product. On the basis of the match between the expected output and the actual output of the system, the software product is deemed to be successful or not. Schach (2008) claimed that this model of software development also prevailed in the 1970's and referred to the model as the "development-then-maintenance" model (p. 9).

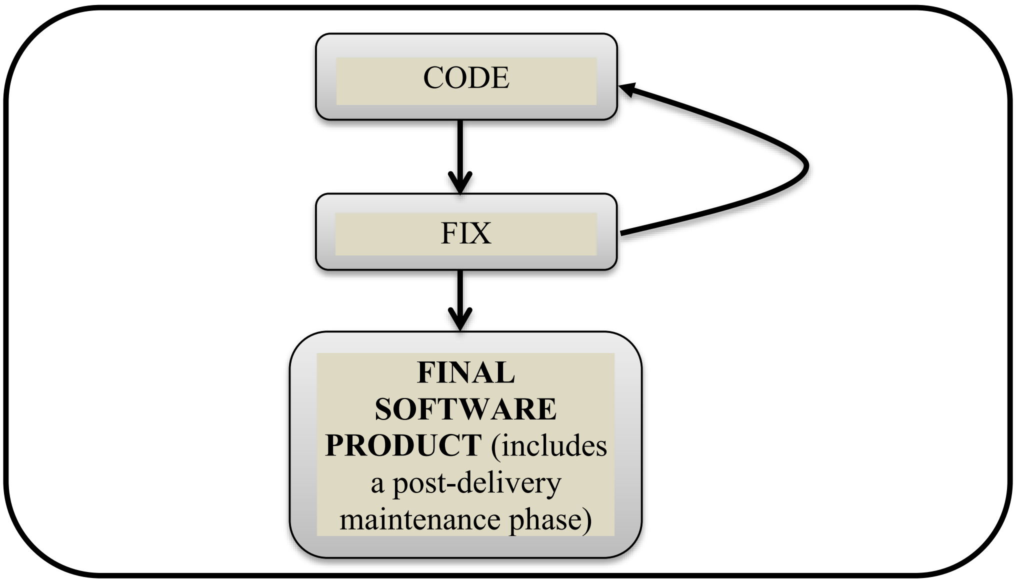

CODE
FIX
FINAL SOFTWARE PRODUCT (includes a post-delivery maintenance phase)

Figure 2.3: Code and Fix Software Process Model (adapted from Schach (2008, p. 50)

The focus of the effort is around the actual coding of the system and the allure of the process is that the software product begins to materialise almost immediately, thereby reducing the “invisibility” (Brooks, 1987, p. 3) of the evolving software product. The ‘code and fix’ SPM basically ‘got the job done’ and it was reported by Trauring (2002) that this unstructured process gave rise to the following criteria that were used to measure the success of a software product.

These criteria were that software:

- should have a relatively low cost of initial development;
- is highly maintainable;
- is portable to different hardware platforms;
- performs the processing expected by the customer.

These criteria epitomised much of what is expected from a software product and became a benchmark for software success that has maintained its relevance over a period of time (Kaur & Sengupta, 2013; Pressman, 2010; Van Veenendaal, 2008). However, while the code and fix process model was functional in producing software quickly, ironically it did not abide by the very same criteria that became a benchmark for software success. Lehman (1980) referred to the code and fix model as one that lacked any guiding theory and made no formal attempt to ensure accuracy or validity of the emergent software product. Boehm (1988) highlighted 3 significant weaknesses of the code and fix model. These weaknesses are listed as:

- A number of fixes contributed to a code base that became difficult to manage; this observation prompted a call for a design phase prior to coding;
- The emergent software product did not meet with the user’s requirements thereby necessitating the need for a requirements phase prior to design;
- The lack of provision for a testing and a maintenance phase resulted in increased costs to modify the software to satisfy user requirements.

Schach (2008) commented that the code and fix SPM may work well for software tasks consisting of less than 200 lines of code. However, it did not scale well for software products that contained higher levels of complexity or delivered substantive functionality. Schach (2008) also concurred with Boehm's criticism of the code and fix model. A common source of concern was the high cost of maintenance incurred as well as the inability to handle changing user requirements. Another area of concern was that the code and fix SPM did not ensure any form of accountability from the actual computer programmers because there were no specifications in terms of what constituted as a successful software product. The ease of code modification gave rise to a "hacker culture" (Boehm, 2006, p. 14) enabling computer programmers to adopt a strategy of hastily patching faulty code to meet project deadlines. This negative indictment on the code and fix model is sustained by Schach (2008, p. 51) who commented that the code and fix model "…is the easiest way to develop software and by far the worst way".

Much of the criticism levelled at code and fix SPM stemmed from a lack of up-front planning and design to underpin the software development effort. In an effort to add elements of planning and design to the software process, Dijkstra introduced his method of structured design at the NATO Conference on Software Engineering Techniques in 1969 (see Dijkstra, 1970).

The structured design method entailed the following main strategies (Jensen, 1981):

- Postpone details – prioritise major functions early and focus on details later;
- Make decisions at each level of abstraction regarding alternate design paths;
- Be flexible – the existing structure design structure should be amenable to change;
- Consider the data at lower levels of abstraction;
- Make an effort to reduce software complexity.

A significant consequence of these software development deliberations was that a paradigm of software development began to emerge. The emergent paradigm of software development consisted of a mix of SPM's and SDM's. The software process model (SPM) provided guidance on 'what needed to be done' while the software development methodology (SDM) provided guidance on 'how to do it'. From a more formal perspective, the SPM provided guidance from an 'elevated' level on the sequence that needs to be followed when moving from one software development phase to the next as well as the criteria that needs to be met in order to sanction progression through the various phases. At the operational level, a SDM consisted of a set of software development methods such as structured analysis and design, stepwise refinement and flowcharting that enabled the attainment of the objectives of each specific phase of the SPM. The relationship between the SPM and the set of techniques, collectively referred to as the SDM is illustrated in Figure 2.4.

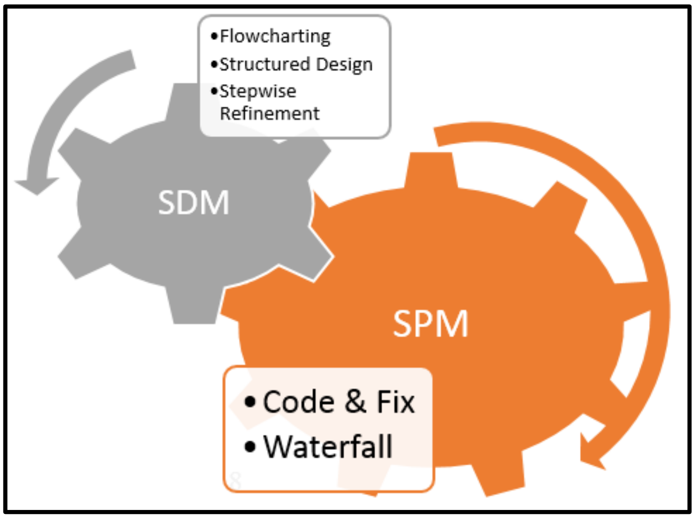


Figure 2.4: The Relationship between the SPM and SDM's

The acceptance by the software practitioner community of a SDM that consisted of techniques such as the structured design method and flowcharting (to a lesser extent) paved the way for a strong focus on requirements analysis and design as precursors to the actual coding phase. In a review of the software development methods that had received much prominence in the late 1960's and early 1970's, Boehm (1988) commented that there was a need for a software process model that guided the software development process through the sequence of stages from analysis to design to coding. The quest for such a well-defined process model was aligned to the SE imperative to adopt an approach for software development that was prescriptive, well defined and resembled a manufacturing process that was similar to the traditional branches of engineering (Mahoney, 2004). This quest for a well-defined, prescriptive and highly controlled SPM was pivotal in promoting the viability of the Waterfall SPM.

2.3.2 The Waterfall Software Process Model

The strategy of adopting a well-defined procedural approach for the development of a software system was first implemented by Benington (1987) as a 9 stage sequential, procedural approach to guide the development of software to control an air defence system for the United States (US) Air Force. In a seminal paper titled 'Managing the development of large software systems' Royce (1970) presented a modified version of the 9-stage sequential model for software development and named it the Waterfall model (the name represented progress of the development process just as water would travel/progress down a waterfall), illustrated in Figure 2.5.

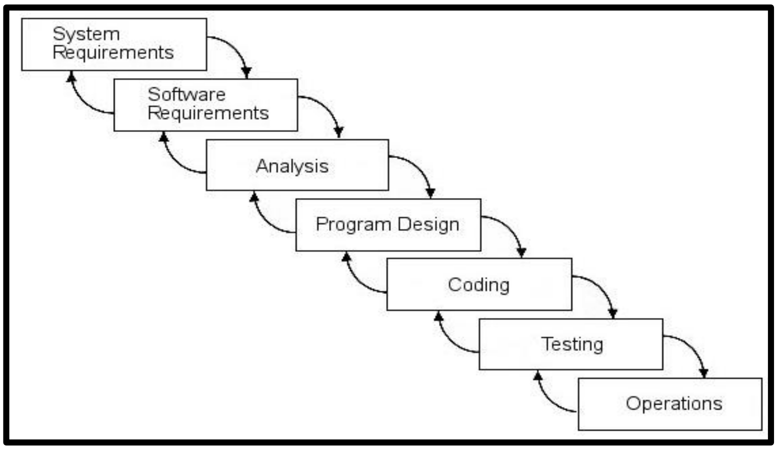


Figure 2.5: The Waterfall Software Process Model (Royce, 1970)

Details of the Waterfall Approach

Royce's Waterfall SPM consisted of 7 sequential steps that contained iterations between preceding and successive steps. However, a major source of contention was the lack of iteration between non-adjacent steps in the model. Royce (1970) did concede that this was a potential weakness of the model and tried to embed more layers of iteration into the model. He proposed execution of the 7 step sequence twice where the first iteration was regarded as a preliminary/prototyped version with the intention of getting to understand the requirements a lot better thereby enhancing the prospect of developing an accurate design model which in turn would arguably ensure that the system would meet its operational expectations. While this iterative intervention boded well for imparting an element of dynamism into the model, it is negated by the model's reliance on substantive documentation requirements that underpinned each stage of development. It is reported in Victor (2003, p. 55) that the allure of the simplicity of the "…single pass, document-driven waterfall model of requirements, design, implementation held sway during the first attempts to create the ideal development process." Hence, the simplicity of the Waterfall model was its biggest advantage and provided the SE community with a solution to the quest to find an orderly, accountable and quantifiable SPM. With the passage of time, there was a growing criticism of the Waterfall model's capacity to handle software requirements that were becoming increasingly complex. These criticisms, summarised in Parnas and Clements (1985), Sommerville (1996) as well as Ranjeeth *et al.* (2013), include the following:

- a system's users seldom know exactly what they want and cannot articulate all they know;
- even if the system's users could state all requirements, there are many details that they can only discover once they are well into implementation;
- Even if the system's users knew all these details, as humans we can master only so much complexity;
- even if the system's users could master all this complexity, external forces lead to changes in requirements some of which may invalidate earlier decisions.

The criticism of the Waterfall model is also alluded to by Nerur *et al.* (2005) who make the observation that the model has a propensity to foster an adversarial relationship between the people involved in the development of the software system, prompting a suggestion that development of software using this approach was not intrinsically rewarding.

These factors coerced the SE community to look at other process models. At an international conference on Systems Analysis and Design held in September 1980 at Georgia State University, McCracken and Jackson (1982) criticised the concept of the systems development life cycle (SDLC). This criticism was based on the commonly held perception that the SDLC was synonymous with the Waterfall approach to systems development. Based on this assumption, the SDLC approach necessitated the early 'freezing' of requirements and a lack of end user involvement in the latter stages of systems development. According to McCracken and Jackson, this strategy (of using a SDLC approach) perpetuates the failure of the SE community to obviate the communication gap between the end user and the systems analyst. McCracken and Jackson proposed 2 alternate approaches to software systems development, both of which do not fit the SDLC mould of development. These approaches are briefly described below:

- Systems development is heavily dependent on end user involvement and presence during all phases of the development process. The development team engages with end users to produce a prototype of the system. Based on feedback from user interaction with the prototype, the development team continually refines the prototype until it eventually evolves into a final product;

- A process of systems development that involves repetition of the following activities: implement, design, specify, re-design and re-implement.

A significant aspect of the software development approach advocated by McCracken and Jackson (1982) was that at the inception of the software development process, a working version of the software system should be made available to the end user and it should form the basis for further refinements until a final system is developed.

The Quest for an Iterative Enhancement Technique

According to Larman and Basili (2003), there were numerous SPM's advocated by the SE community and a 'common theme' in all these process models was the deliberate effort made to avoid a single-pass, sequential, document-driven approach. The common thread in these deliberations regarding software process improvement (SPI) initiatives was that software development process should follow an 'iterative enhancement' technique as suggested by McCracken and Jackson. The objective of these SPI initiatives was to minimise the "…gulf that exists between the user and the developer perspectives on a system" (Reid Turner *et al.*, 1999, p. 3). According to Reid Turner *et al.* (1999), the user perspective of the system is centred in the problem or business domain while the developer's focus is on the creation and maintenance of software artefacts that represent the developer's interpretation of the problem domain. It does not necessarily reflect the reality as experienced by the end user when using the system in the actual problem or business domain. In order to acquire this realistic view of the system, the developer has to iteratively expose the end user to incremental views of the evolving system so that the feedback obtained can be used to underpin all phases of the development process.

Ideally, a SPM should enhance the prospect of accurately predicting user requirements (Davis *et al.*, 1988). While this objective may not be fully achieved, it is imperative that SPM's should be engineered to obviate the gap between the user and developer perspectives of a software system. In this respect, the iterative and incremental approach is more compatible with the philosophy of sustaining a focus on user requirements rather than focusing on the operational aspects of the software development process as is embodied by the Waterfall SPM.

2.3.3 Iterative and Incremental Software Process Models

Basili and Turner (1975) suggested that a software process model that consists of an iterative and incremental development (IID) methodology would entail a "…sequence of successive design and implementation steps, beginning with an initial 'guess' design and implementation of a skeletal sub-problem" (p. 395). After successive iterations that continuously elicit end user feedback, the system's design model is refined and the solution to the initial skeletal sub-problem is evolved into a complete solution to the actual problem. This approach embodies a more accurate modelling of the business/problem domain thereby enhancing the maintainability[3] and the robustness[4] of the final software product. Hence, any SPM based on IID methodology would have to start with the implementation of a subset of the system requirements and incrementally build functionality onto the evolving system until the final product is developed. An IID software process model, referred to as the "antithesis" of the Waterfall SPM (Booch, 1986, p. 232), had been employed in numerous software projects in the 1960's and 1970's (Larman & Basili, 2003). This strategy was formally presented in a seminal publication by Basili and Turner titled "Iterative Enhancement: A Practical Technique for Software Development" (see Basili & Turner, 1975). A significant deviation of the IID methodology from previous SPM's and SDM's is that IID does not impose the condition of having all the system requirements being declared 'up front'. Larman and Basili make reference to a set of core requirements that are listed as tasks in a "project control list" (p. 390) that is refined while the system is being developed. The project control list acts as a project management instrument that provides an indicator of the progress made with meeting system requirements. Each task in the project control list becomes the subject of the iterative activities of analysis, design and implementation. The project control list is an inherently dynamic list that is refined on the basis of increased knowledge that the development team acquires with regards to the system requirements. This knowledge is obtained by virtue of feedback from end user interaction with the earlier, incremental versions of the system (Larman & Basili, 2003). From a SE perspective, the strategy of keeping the systems developers as well informed as

[3] Software maintenance is the activity performed whenever a fault is fixed or an adjustment is made to the software product to accommodate a change.in the set of requirements (Schach, 2008, p. 11),

[4] The ability of a software product to accommodate changes without any degradation in performance of the software product (Schach, 2008, p. 47)

possible at all stages of development is a compulsory requirement for ensuring quality of the final system (Mills, 1980). Also, the heavy reliance on end-user involvement at all stages of development is aligned to the prototyping approach suggested by McCracken and Jackson (1982).

This iterative approach to software development represented a significant change to the sequential Waterfall-like approach and formed the basis of a new paradigm[5] of software development. The focus of the development effort is on refining user requirements, not confining them to a pre-defined prescriptive list of requirements. In terms of Brooks' quest to find the elusive 'silver bullet'[6] that will obviate some of the difficulties associated with software development, Brooks' comments with regards to the IID methodology are summarised below:

- One of the important activities performed by a software developer is the accurate extraction and refinement of the requirements for a software system. Most often, the client is not fully aware of the requirements themselves, hence there has to be extensive iteration between the client and the system developer so that an accurate idea of the system requirements is obtained;
- Even with extensive consultation between the client and the systems developer, it is still difficult for the client to provide a precise specification for the software system;
- One of the most promising technological developments that represents a viable 'attack' on the essence of software complexity is the rapid prototyping approach that is part of the iterative specification of requirements;
- Software systems should be 'grown' or developed using an incremental, evolutionary approach. It should not be 'built' from an initial set of prescribed requirements. This is aligned to the strategy proposed in Mills (1980) that a software system should first be made

[5] The sequential Waterfall SPM represented the older paradigm of software development.

[6] See Brooks (1987) for an elaboration of the "silver bullet" analogy

> to run successfully, even if it did not do anything meaningful. The system could simply demonstrate its ability to successfully activate high level 'dummy subprograms'. Hereafter, the 'dummy subprograms' should be developed using a method of stepwise refinement (Wirth, 1971) until the lower level subprograms are populated with actual program code.

Brooks (1987) concludes by remarking that IID methodology has had a profound impact on the effectiveness of software development process models.

What has emerged from the preceding discussion is that the IID methodology allows the developer an opportunity to refine the system requirements on the basis of responses obtained from end users' interaction with a working version of the actual system. The system is also 'grown' incrementally into the final product. The IID methodology was proposed as a mechanism to divert the sequential mentality inherent in the Waterfall SPM to a more dynamic one. However, the individual phases of the Waterfall SPM form the core elements of the IID based SPM's as illustrated in Figure 2.6.

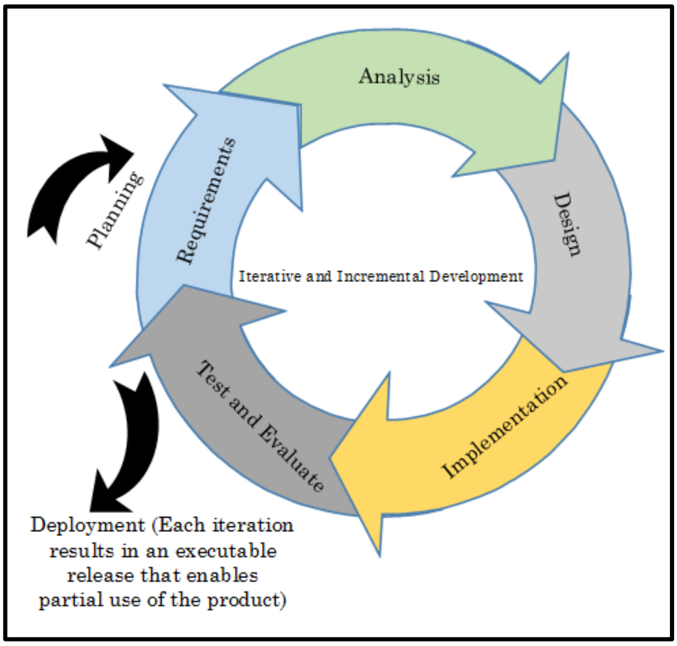

Analysis
Design
Implementation
Test and Evaluate
Requirements
Planning
Iterative and Incremental Development
Deployment (Each iteration results in an executable release that enables partial use of the product)

Figure 2.6: IID based Software Process Model (adapted from IBM (1998))

Whilst the methodological aspects of the iterative and incremental approach are quite clear (as illustrated in Figure 2.6), a variety of SPM's based on the IID methodology were proposed by the SE community. The essence of these models is that they need to have a mechanism that enables developers to test the system with end users in order to refine the system's design models during the development process (Cusumano & Selby, 1997). A discussion of SPM's that incorporate the IID methodology is presented in the subsequent text.

2.3.4 The Spiral Software Process Model

In a paper titled 'A Spiral Model of Software Development and Enhancement' Boehm (1988) presented a modification of the Waterfall SPM by introducing a SPM based on iterative and incremental development methodology that was named the Spiral Model of software development. Boehm contextualised the Spiral SPM by suggesting that the generic IID based SPM could easily degenerate into a code and fix style of development. According to Boehm (1988), software developers who develop with an IID mentality would be inclined to prioritise changing end user requirements without being constrained by an overall planning and risk mitigation strategy. Such an approach would have a strong propensity to produce 'spaghetti' code that would render software systems as unmaintainable. In response to this perceived weakness of the IID SPM, Boehm's Spiral model incorporated elements of the IID methodology within a framework that included a liberal presence of planning and risk mitigation initiatives in each of the iterative development cycles as illustrated in Figure 2.7.

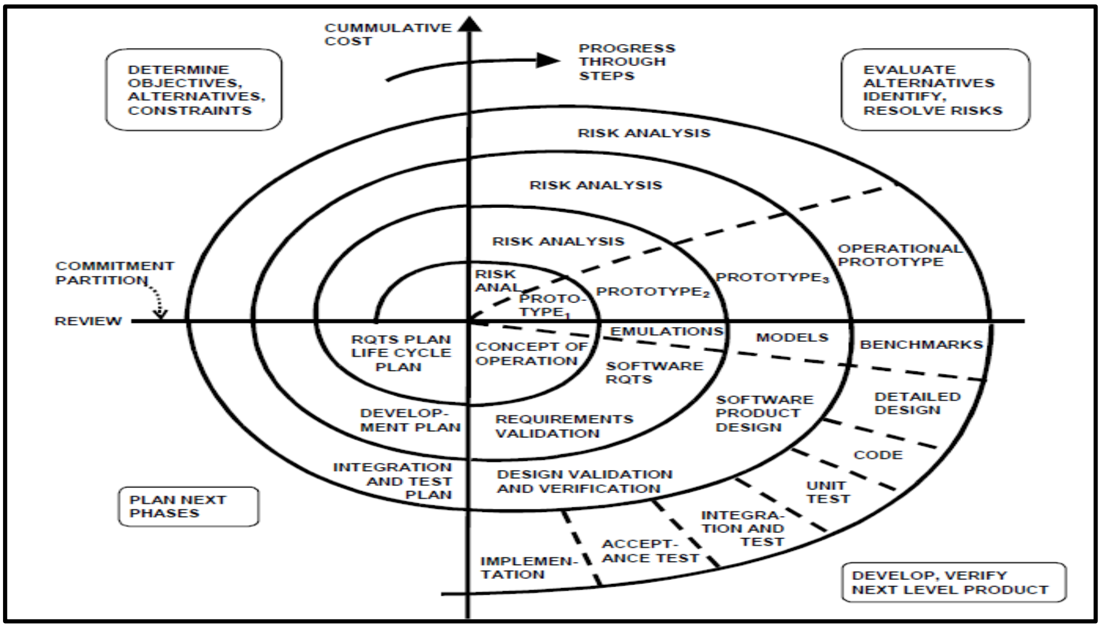


Figure 2.7: The Spiral Software Process Model (Boehm, 1988)

As can be seen in Figure 2.7, the Spiral SPM entails an iterative sequence of the following activities for each specific portion of the system.

- Identification of objectives, alternatives and constraints for specific phases of the development cycle (upper left quadrant in Figure 2.7). The objectives of a specific phase of development entails aspects such as the desired functionality, the expected performance of the system, the ability to accommodate change, etc. (Boehm, 1988). The alternatives allude to identification of possible alternative designs and off-the-shelf solutions. The constraints refer to operational parameters that are normally expected as part of the systems development process. This includes cost, resource and interface constraints;
- An evaluation of the alternatives (identified earlier) relative to the constraints and objectives. A proof-of-concept prototype[7] is

[7] A proof-of-concept prototype is a scale model constructed to test the feasibility of construction (Schach, 2008, p. 61)

suggested as a possible risk evaluation strategy. These activities form the upper right quadrant of the Spiral model;

- The lower right quadrant of the Spiral SPM represents the development phase of the model. However, the significant aspect of the development phase is its dynamic nature. If concerns regarding the system performance or the viability of the user interface cannot be resolved, then an evolutionary[8] development approached is suggested. In this case, the system is developed incrementally. The main aspects of the system are developed early, thereby providing the developers with an opportunity to evaluate the risks of system failure at an early stage of development. The system is evolved into the fully, fledged final product. If, however, all performance and user interface risks have been identified and resolved at the requirements phase, then the basic waterfall approach to software development may be followed (as indicated in the lower right quadrant). This is not the same as simply a different case of the Waterfall SPM. Boehm makes reference to a software development strategy that entails partitioning of the software product into components that are developed iteratively using the phases of the Waterfall SPM.

Spiral is a Typical IID Methodology

Based on the illustration and the subsequent narrative, the Spiral SPM is representative of a typical IID methodology. However, according to Schach (2008, p. 64) the Spiral model is not a “truly incremental model” because it consists of discrete phases of “waterfall-like” development (in reference to the different quadrants that underpin the Spiral SPM as illustrated in Figure 2.7).This assertion is certainly debatable because the strategy of prototyping ensures incremental refinement of system requirements, unlike the Waterfall approach where the requirements are declared and ‘frozen’ at the beginning of the

[8] The evolutionary approach is described by Pressman (2010, p. 42) as an approach that produces an increasingly more complete version of the software with each iteration

development process. Boehm and Hansen (2000) are in agreement with the dynamism inherent in the Spiral SPM by virtue of their assertion that the Spiral model embodies a cyclic approach where the objective is to incrementally refine a system's degree of definition whilst at the same time, reducing the degree of risk. Ruparelia (2010) endorses the suggestion regarding the incremental nature of the Spiral SPM by claiming that the philosophy underlying the spiral approach is to start small, and think big. This is certainly aligned to the IID methodology of focusing development on small, manageable portions of the system. It is expected that each portion will be aligned to the overall system development objectives that attempt to ensure optimal functionality without incurring significant cost overheads and also ensuring the maintainability of the system. Hence, whilst there is a case to be made for the Spiral SPM to be regarded as an iterative and incremental type SPM, the "quadrant-like" structure of the Spiral model (Figure 2.7) does give credibility to Schach's interpretation that the Spiral SPM simply entails successive iterations of the phases of the Waterfall SPM. Hence, a closer inspection of the Spiral SPM is warranted in order to resolve the doubt cast by Schach regarding the iterative and incremental nature of the Spiral SPM.

The problem of creating a SPM that attempts to minimise the weaknesses of other SPM's is that the complexity of the newly created process model increases. This assertion may be substantiated by using the case of risk analysis in the Spiral SPM. Risk analysis was included as a control measure in the Spiral SPM so that the incremental functionality of the evolving system was added within the parameters of cost and resource consumption feasibility. However, the stringent implementation of such control measures necessitates the introduction of a new set of processes and activities that will require concise and comprehensive documentation in order to create an effective 'audit trail' of the risk analysis deliberations. This added overhead to the Spiral SPM tends to render the process as a 'high ceremony' process.

Spiral Methodology viewed as a "High in Ceremony" Process

The expression 'high ceremony' was coined in the annals of SE literature to refer to SPM's that entailed substantive focus on documentation, formal software reviews, rigid adherence to methodology and embodied a highly controlled demeanour towards the software development process. The attributes of a high ceremony process have been gleaned from the use of the expression by McBreen (2000), McCormick (2001) and Booch (2001) in reference to general software development methodology as well as Cockburn (1999), Fowler (2001) and Fowler (2006) in reference to agile software development methodology. From the preceding references, the 'waterfall-like' approach to software development is regarded as high ceremony while the SPM's that are strongly influenced by an iterative and incremental approach and gives prominence to the visibility of the software rather than the documentation, is regarded as 'low ceremony'.

A distinctive aspect of the Spiral SPM is that it has elements of the IID methodology, although the major focus is on extensive risk analysis. The risk analysis overhead renders the Spiral approach to software development as a high ceremony SPM. As such, it is ideally suited for large-scale software systems that serve a 'mission critical', institutionalised purpose and entails a substantive financial and resource investment. In such systems, the cost of failure warrants an approach that is underpinned by continuous risk analysis, thereby minimising the prospect of system failure and enhancing the traceability and accountability of the software development process. The Spiral SPM embodies an engineering-like approach to software development, the significance of which is highlighted in Booch's commentary on the future of SPM's, where he refers to the idealism of "…managing requirements, iteratively and incrementally growing a system's architecture, controlling change, and testing continuously" (Booch, 2001, p. 120). A significant aspect of the preceding comment is that Booch refers to the idealism of incorporating proper software development principles into the process of software development. The implication here is that the ideal route is perceived to be more of a vision of perfection rather than being representative of what is practically feasible. In this regard, the Spiral SPM is theoretically sound, but practically untenable.

Given the rigour and effort spent on ensuring system success, there is a lack of popularity of the Spiral SPM. While the SE community has endorsed the Spiral SPM as a viable approach to software development, it is the economic imperatives that have 'derailed' any prospect of a unanimous show of support for the Spiral SPM. Booch (2001) points out that economic constraints tend to be given priority over quality software thereby resulting in SPM's that produce software systems that are less than optimal. In an article titled, "When Good Enough Software is Best", (Yourdon, 1995) coined the expression *Good Enough Software*. In this article Yourdon asserts that software development does not have to always abide by the rigour, standards and precision of engineering-like projects. He contends that a "good-enough approach" (p. 79) would faciltate a software development process that is rational and attaches significance to the domain of usage of the software system thereby modulating the exclusive focus on the rigour and precision of the development process. Meyer (2003) endorsed the concept of *good enough software* and suggested that a software system with 'good enough' quality, that is delivered on time so as to enhance the prospect of competitive business advantage is better than a software system that is deemed to be perfect, but is delivered too late to provide business value. It is within this context that high ceremony SPM's such as the Spiral model began to lose popularity amongst software developers.

Schach (2008) pointed some other issues of concern regarding the Spiral SPM. Schach contends that the additional costs incurred by the risk analysis phases of the Spiral SPM renders the Spiral approach appropriate for large-scale software projects where the cost of the risk analysis is only a small percentage of the entire project. He also refers to the skill required by the analysis team in reliably identifying areas of risk. If this is not done accurately, then the cost of recovery from such risks could be quite substantial. Boehm (1988) has acknowledged the criticisms of the Spiral SPM by members of the SE community and has conceded that the model relies heavily on comprehensive documentation and the involvement of highly experienced software practitioners who are skilled in risk analysis.

Conditions for the Successful Implementation of the Spiral Methodology

From the prevailing discussion, a necessary condition for the successful implementation of the Spiral model is to always have a multi-skilled team available. While software development skills are an absolute necessity, the development team also needs to have the expertise of experienced software developers readily available to make risk assessments during the life-cycle of the Spiral model. The main issue with these requirements is that the intensity of demands for software availability cannot be satisfied by software developers who always take the high ceremony approach. For most business oriented systems that serve an immediate need, a satisficing approach that entails the development of 'good enough' software may be perceived to be a much more viable alternative to the Spiral SPM. Naumann and Jenkins (1982) commented that SPM's need to be adjusted to handle software requirements that are more complex and operate in less structured environments. These SPM's need to be able to handle changing user requirements, respond to the feedback provided on the basis of users' experiences in interacting with initial versions of the system as well as be able to handle new technology. Hence, there was a call for SPM's to become more dynamic

The Spiral SPM had served its purpose in contributing to the evolutionary trajectory of SPM's. However, the potential of the Spiral SPM to be accommodating of a dynamic development environment was limited. This quest for a dynamic software process model that was flexible enough to handle changing user requirements became an urgent source of enquiry by the SE community. Aligned to the issue of obtaining an accurate and adaptable representation of user requirements, the object-oriented (OO) paradigm of software development was beginning to achieve a ubiquitous presence in the field of software development. According to Capretz (2003), the OO paradigm enhanced the prospect of obtaining a better software based representation of real world artefacts. The Unified Process (UP) is a SPM that leverages the advantages inherent in OO development to offer a truly IID methodology.

2.3.5 The Unified Process

The UP is representative of an iterative and incremental methodology where each increment entails all of the activities of the traditional SDLC. However, the significant aspect of the UP is that each iteration is not an independent activity and according to Jacobson *et al.* (1999), each increment yields a working software artefact that provides the client/user with an early opportunity to interact with the evolving system in order to provide feedback regarding expectations of the system. The UP conforms to the generic framework inherent in most software process models. According to Pressman (2010) the generic framework of software development consists of the activities of requirements gathering, system planning, analysis and design, coding, testing and deployment. These activities are manifested in the UP in an iterative manner, as depicted in Figure 2.8 below. The iterative demeanour of the UP is required to accommodate changes to the evolving system based on user feedback. This is a pivotal area of differentiation between the Waterfall-like software process models and the UP. As illustrated in Figure 2.8, there is a substantial overlap between the different phases of the UP. The UP is architecturally modelled on the phases that underpin a typical SDLC. The iterative and incremental features are incorporated by virtue of the UP's two-dimensional structure. The phases of the UP are illustrated in Figure 2.8.

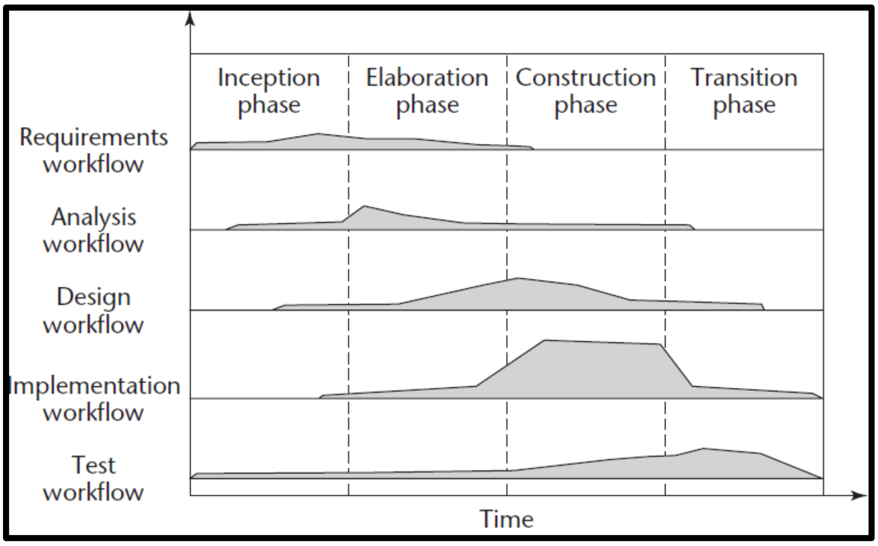


Figure 2.8: The Unified Process (Schach, 2008, p. 86)

The ***Inception Phase*** is predominantly focused on establishing the system's objectives and the economic viability of developing the system. This phase is regarded as a risk analysis phase where a business case for the development of the system is compiled. The ***Elaboration Phase*** entails refinement of the system's objectives, the compilation of a software project management plan and the development of system models that provide an abstraction of the complete system. Unified Modelling Language (UML) is used to represent these models. The ***Construction Phase*** is predominantly focused on coding and testing activities. The first fully operational version of the system is released as a beta release where feedback is obtained from the client regarding expectations of the system. The iterative process of correcting system faults so that it conforms to the client's expectations is regarded as the ***Transition Phase.***

It should be noted that the phases are not mutually exclusive as can be observed in the illustration in Figure 2.8. Schach (2008) makes a point of explaining that the four phases of development depicted in the illustration of the UP are representative of a typical software development effort where functionality is added to the evolving system in an incremental manner. The UP represented a software process model that incorporated much of the flexibility/agility of an iterative and incremental approach to software development. It heralded the beginning of an era that embraced an adaptive rather than a prescriptive approach to software development.

However, the UP was still driven by a substantive design effort prior to intensive coding. The elaboration and construction phases are heavily reliant on extensive UML based models that needed to be coupled with substantial documentation. According to Fowler (2001) this situation does not auger well for software projects where the requirements are volatile and not clearly understood. Fowler suggests that there is a tendency for the UP to follow a 'big design up front' (BDUF) approach, thereby compromising the ability of the software development team to handle new or changing requirements paradoxically leading to a situation where the UP degenerates into a waterfall-like approach. The BDUF intrinsic to the UP has also been criticised for creating a situation where the prevalence of a complex inheritance hierarchy[9] compromises the maintainability and extensibility of the system (Bennett & Rajlich, 2000). As a response to this situation, Ambler (2001) suggested invocation of an iterative approach to modelling as opposed to adopting a BDUF approach. This can be achieved by creating high level, lightweight models such as use case diagrams and user stories that are not necessarily specified to completion. These so called agile models become the basis for coding efforts that result in the incremental release of versions of the software product that are refined iteratively until it meets with user expectation. This strategy is commensurate with the ideas previously expressed by Fowler (2000) in an article titled "The New Methodology" where he alludes to the strategy of integrating design and coding, thereby blurring the distinction between these two phases of the software process.

This iterative relationship between the design and coding phases is essential in order to accommodate the volatile nature of software requirements. From a UP perspective, the consequence of such a strategy is the conflation of the elaboration and construction phases of the UP, thereby prioritising working software over comprehensive documentation. Fowler extends this idea by suggesting that the source code is actually an integral part of the documentation. These deliberations converge towards a software process model that is underpinned by the structure of the UP and supported with a set of agile models. However, in order for such a software process model to succeed, Ambler (2001) warns that an organisation's culture should be receptive to the UP and agile modelling. Ambler makes the point that the UP is normally adopted by organisations that are 'documentation centric' and tend to align themselves with a development philosophy that is fairly rigid and prescriptive. In such a situation, an agile approach to modelling is not easily reconciled with the prevailing prescriptive mind set. In 2001, Ambler proposed a software process model named the Agile Unified Process (AUP) (see Ambler (2001)) that consisted of a lightweight version of the UP.

[9] Inheritance is an OO strategy to enable code reusability

2.3.6 The Agile Unified Process (AUP)

The AUP preserves some of the formality of the UP but it also consists of an agile ‘flavour’ that is underpinned by lightweight models and frequent software releases. The evolving software artefact is used as a validation instrument rather than the documentation or the system’s analysis and design models. The elevated priority attached to the actual coding of the software system coupled with an ideology that UML and other non-code artefacts are of secondary importance, prompted the software engineering community to develop a software process model that is receptive to changing user requirements. The focus of the development effort shifted from the subservience to the process models that dictated software development to a higher level of interactivity with the end users of the system. This embodied a shift in focus to the social context in which software systems were being used. According to Boehm (2006), this shift in emphasis to a software process model that enhanced the prospect of rapid software development is necessitated by societal demands where software is being used as a “competitive discriminator” (p. 18). The increased pressure on the software development community to ‘reduce software time-to-market’, prompted the advent of software process models that are underpinned by characteristics that are iterative, incremental and agile. The renewed focus on extensive client collaboration, less intensive modelling, quick coding and instantiation of software artefacts to enable client validation, converges to the idea that adoption of a specific SPM may be perceived as too restrictive in achieving these objectives. This assertion resonates with the claim made by Nandhakumar and Avison (1999) that a SPM has an invisible presence that merely serves as a controlling rather than a usable framework that adds value to the software development process. This remark may have been a catalyst for the formalisation of a set of software development methods that embody an element of dynamism and lightweight structure to the software development process. Collectively, these methods are referred to as agile methodology.

2.4 Agile Software Development

According to Cohen et al. (2004), an agile approach to software development is a reaction to the traditional ways of developing software and an acknowledgement of the "…need for an alternative to documentation driven, heavyweight software development processes" (p. 3). Cockburn (2002) stresses that the focal point of agile methods is to facilitate a software process model that embodies flexibility and the capacity to handle changing requirements. According to Abbas *et al.* (2008) these elements of agility have been intrinsic to the software development process for a number of years prior to the formalization of the methodology in the late 1990s. Cohen *et al.* (2004) suggested that agile characteristics such as the response to change, customer involvement, and working software in preference to elaborate documentation became increasingly important to the software development process from 1975 onwards. These attributes of the software development process were collectively referred to as the iterative enhancement development technique. However, these attributes were being used in a fragmented manner without being recognized as part of an overriding software development methodology.

2.4.1 The Need for an Agile Intervention

The preceding discussion on software development process models and methodologies that culminated in the UP (sections 2.3.1 to 2.3.6), seem to suggest that there is a lingering element of doubt and discontent by software practitioners regarding the effectiveness of ad hoc techniques and prescriptive process models in solving the problem of successful software development. This assertion is corroborated by Strode (2012) who suggests that in the late 1990's, the IS development research community seems to converge to an opinion that software development *methodology* is not universally beneficial and in some instances, it can be quite detrimental to the efforts of developing software successfully. Despite the best efforts of the software engineering and IS research communities, the software development process models have not achieved a level of success that would arguably ensure confidence in the successful development of a software artefact. This is contrary to other engineering disciplines where a planned, controlled, engineering-like approach to development would normally guarantee a successful product. The preceding assertion is commensurate with the overriding opinion that emerged from the 1996 seminar titled "The History of Software Engineering" (see Aspray *et al.*, 1997) and is also partially aligned to the call by Fenton *et al.* (1997) for the adoption of a more scientific approach to the software development process.

Tackling the SE Crisis

There was a need for the software engineering community to re-establish a measure of confidence in the software development process. An impending crisis situation such as the one that confronted the software engineering community is normally tackled by examining the epistemological underpinnings of a discipline. Incursions into the epistemological underpinnings of the computing discipline can be quite diverse and could easily degenerate into a problem in itself that may well 'creep' beyond the scope of the current discourse. However the ontological synopsis of the discipline of computing by Jackson (1995) in an article titled "The World and the Machine", provides a frame of reference from which there may be a generation of some discussion on the topic. Jackson undertakes an exploration of the relationship between the 'world' (a reference to society) and a machine (a reference to a computing device). According to Jackson, this relationship has at least four facets. These are:

- the *modelling* facet where the computing device simulates society;
- the *interface* facet that provides society with a mechanism to interact with the computing device;
- the *engineering* facet, where the computing device exerts a degree of control over the societal behavioural patterns;
- the *problem* facet where societal convention influences the shape/form of the computing device and the solution that it provides.

The preceding lucid illustration of what could potentially be a complex relationship paves the way for a high level analysis of the relationship between society and a computing device. Jackson identifies the conceptual gaps of understanding between the technological realm and the societal realm as possibly an area where the software engineering community has failed society. This conceptual gap of understanding manifests in all four facets of the relationship between the world and the machine.

The Social Dimension of SE

While substantial progress has been made in providing structure and guidance for the technical aspects of software development, not much has been done to address the communication gap between the software engineering community and the society of human end-users. Jackson concludes this ontological analysis on the state of computing by suggesting that discourse on software development methodology should focus on the human realm and the development of methodologies and process models that elevate the importance of the social element in the software development process. The social dimension has a strong presence in the requirements elicitation phase, the modelling and coding phases that typically involve a team of developers and finally within the context of use that has an impact on end user behaviour at the organisational level. These socially oriented issues were not as prominent in the early 1990's where software development typically involved a small group of end users and developers who interacted with standalone, "stovepipe" systems (Boehm, 2006, p. 23) that had a limited potential to enable interoperability with other systems and exerted a degree of influence that was seldom felt at the organisational level. The late 1990's and early 2000's heralded a change in this situation. Software began to play a more pivotal role in society thereby becoming a catalyst for the software engineering community to explore software process models that endeavoured to incorporate processes that embodied an acknowledgement of the human influence on the software development process. This assertion is corroborated in Highsmith and Cockburn (2001) and Cohen *et al.* (2004) who claimed that there was a need for new software development practices that were people-oriented and flexible and are based on generative rules that do not break down in a dynamic environment. Abbas *et al.* (2008) added to the impetus for a more dynamic, agile software process model by suggesting that the traditional methodologies did not adequately cope with the "…turbulence of business demands and fluctuating advances in technology" (p. 3). These unpredictable traits of a changing, modern society rendered it almost impossible to anticipate a complete set of the requirements early in the project lifecycle.

2.4.2 Agile Software Development Methodology

The agile philosophy is centered on the idea of being adaptive and non-prescriptive. Incorporation of this philosophy into the domain of software development culminated in the spawning of software development methods that were grouped together to constitute agile software development methodology. According to Abbas et al. (2008), it is not easy to provide a succinct definition for the concept of agile software development methodology. However, there were contributions regarding some of the defining characteristics of agile software development methods (listed in Table 2.1) that collectively provide an overall illustration of the methodology.

Table 2.1: Characteristics of Agile Software Development Methods

Characteristics	Reference
Lightweight and manoeuvrable; produce the first working software version in a short time frame; invent simple solutions, so there is less to change and making those changes is easier; improve design quality continually	(Highsmith & Cockburn, 2001)
Employs a simple design with short iteration cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation	(Boehm, 2002)
Refactoring, iterative feature-driven cycles, with involvement of customer focus groups	(Highsmith, 2002)

Prioritising project manoeuvrability with respect to shifting requirements, shifting technology, and a shifting understanding of the situation.	(Cockburn, 2002)
Emergent, iterative and exploratory; not confined by formal rules; learning through experimentation and introspection, constantly reframing of the problem and its solution	(Dybâ & Dingsoyr, 2009)
Adaptive, iterative, incremental, and people oriented	(Abbas *et al.*, 2008)

As indicated in Table 2.1, the elements of dynamism, simple design and quick delivery of working software that underpinned most of the agile methods was a reason for the software engineering community to feel upbeat about the prospect of obviating the dilemma regarding the changeability and invisibility[10] that plagued the software development community. The strong focus on providing the customer with a quick, working, initial version of the software system as well as the iterative nature of development provided a measure of confidence that changing customer requirements could easily be accommodated. However, according to Boehm and Turner (2003), agile methods flounder on handling complexity and to some extent conformity. They claim that agile methods are suitable for small projects where there is less complexity. Also, the dynamism inherent in agile methods do not auger well for the desire to impart obedience and order to the software development process. These statements are a serious indictment on the prospect of the newly introduced agile methodology to achieve success levels that could match up to the hype and enthusiasm that these methods initially generated. However, before conducting a critique of the preceding statement, it is only fitting that an incursion into the advent and formalisation of agile methods be undertaken in order to obtain a deeper insight into the philosophical and operational aspects of agile methodology.

[10] A reference to Fred Brooks' indictment on the software crisis in Brooks (1987)

According to Abbas *et al.* (2008) as well as Dingsøyr *et al.* (2012), the term agile methodology is used to collectively refer to lightweight software development methods such as Extreme Programming, Scrum, Dynamic Systems Development Method (DSDM), Feature Driven Development (FDD), Crystal, and Adaptive Software Development (ASD). Each of these methods is centered on core principles listed in Table 2.2. In an effort to consolidate the principles espoused by each of the agile methods listed above, a group of 17 software practitioners, who were instrumental in promoting the use of lightweight software development methods in the late 1990's, put together the *Manifesto for Agile Software Development* (see Fowler & Highsmith, 2001) that documented their shared philosophy of software development (Misra *et al.*, 2012). The Agile Manifesto, consisting of a set of 12 principles, is holistically based on the core principles listed in Table 2.2. The divergence from traditional software development methodology, interpreted from the original set of 12 principles, is also presented in Table 2.2.

Table 2.2: Core Principles of Agile Manifesto and Divergence from Tradition

Core Principles of Agile Manifesto	Divergence from Traditional Practice
Preference is given to individuals and interactions over processes and tools.	An inclination towards a non-prescriptive methodology that is responsive to the social dynamics of the development environment rather than a process; system development is driven by a "self-organising" software development team.
Working software is prioritized over comprehensive documentation	Software is developed incrementally over shorter time scales using smaller designs; The focus is on developing specific features of the system thereby facilitating customer collaboration; working software is regarded as the primary measure of progress.
Customer collaboration is valued more than contract negotiation	System developers maintain a high level of interactivity with the business stakeholders.

Responding to change over following a plan	The lightweight development demeanour enhances the prospect of accommodating changing requirements even late in the development cycle. The rationale is to enable change in order to provide the customer with a competitive advantage.

Cohn, one of the contributors to the Agile Manifesto, makes reference (see Cohn, 2004) to the software development process model that prevailed during the mid-1990 as a 'mix' of the following techniques:

- Extensive collaboration with end users culminating in informal documents that captured the essence of what the end user desired in the system
- Sketching of screen interfaces on paper;
- Prototyping;
- Coding small parts of the system that would be demonstrated to a representative set of end users.

Cohn (2004) claimed that extensive upfront requirements gathering and documentation can be counter-productive. He cited the inaccuracies of the English language as a pivotal aspect that could compromise efforts at capturing accurate user requirements of a system. These sentiments are an endorsement of the philosophy of maintaining a lightweight, adaptive approach to software development that is enshrined in the Agile Manifesto. Cohn suggests that the agile oriented practice of capturing user requirements as a set of user stories, which entails a short description of the required functionality from the perspective of the user or the customer of the software (Cohn, 2004) is more effective in bridging the gap between the end user and the developer. The technique of documenting user requirements as a set of user stories is more closely aligned to agile methodology in contrast with the technique of compiling a comprehensive user requirements document used for prescriptive process models such as the Waterfall approach or the technique of use case modelling, intrinsic to the UP. The point of departure regarding these requirements documentation techniques is that user stories are lightweight in the sense that it captures a minimal set of requirements that become the focus of a single iteration of an agile based software project. In order to contextualise the iterative techniques used by agile methods, an overview discussion of these methods will be presented in the subsequent sections. The discussion will be structured around the listing of agile methods presented in Abbas *et al.* (2008) and Dingsøyr *et al.* (2012).

2.4.3 Extreme Programming (XP) Methodology

Extreme Programming (XP) is a software development methodology that is considered to be the catalyst responsible for generating a focus on ASDM (Fowler, 2013). During the initial period of engagement with ASDM, XP was one of the most commonly used agile methods (Hummel, 2014; Sinha & Prajapati, 2014; van Valkenhoef *et al.*, 2011)

Many of the values and principles of XP, which are documented in Beck (1999) and his publication titled “Extreme Programming Explained” (see Beck, 2000), that reached seminal status, is closely aligned to the values and principles enshrined in the Agile Manifesto. According to Beck (1999), XP transforms the conventional software process models into a sideways orientation as illustrated in Figure 2.9. Beck suggests that the evolution of software process models was structured along the philosophy that reflected a preference for a shorter development cycle because it is conducive to accommodating changing user requirements. This philosophy is reflected in Figure 2.9 where the transition from the Waterfall model and its long development cycles to the shorter iterative development approach epitomised by the Spiral model, culminating in XP where all of the development activities (analysis, design, implementation and test) are ‘blended’ into smaller iterations, throughout the entire software development process.

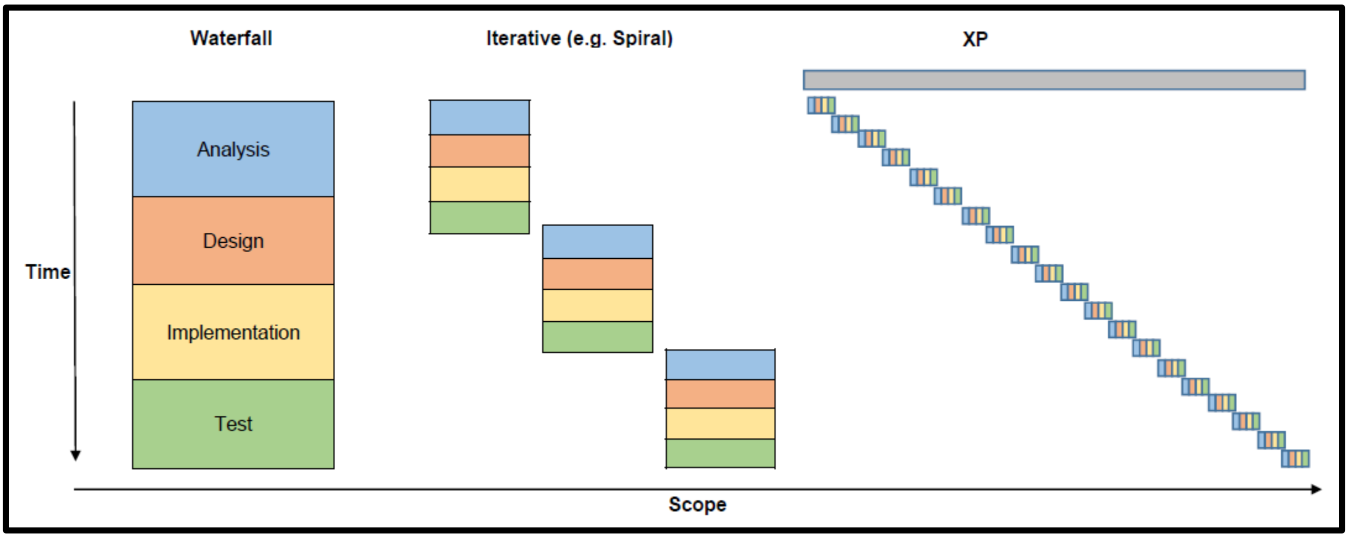


Figure 2.9: The Evolution to XP (Beck, 1999)

Beck's structuring of XP entails a transformation of the conventional software process into a 'sideways' orientation, where the focus is prioritisation of quick coding and testing rather than developing for the future. In order to achieve this strategy of blending the software development activities into smaller iterations, Beck proposed a set of major practices that have to be followed to facilitate compliance with XP methodology. The main practices of the XP methodology are summarised in Table 2.3.

Table 2.3: Core Principles of XP (Adapted from (Beck, 1999))

Planning game	Customers determine the most valuable features that they want prioritised; these features are documented as user stories that contain specifications regarding the scope and timing of the release of the feature; each feature release is regarded as an iteration/*small release* of the XP process model; The planning game is a subtle reference to the interactivity between the "business people"/customer and the "technical people"/programmer.
Metaphor	Each project is guided by a single overarching metaphor/ a story that provides a user friendly/non-technical reference for the basic elements of the system; a piece of system jargon that enables all system stakeholders to identify with the overall purpose of the system; a deliberate attempt to avoid

	a reference to technically oriented terminology such as system architecture (Grinyer, 2007)
Simple Design and refactoring	There is no big design up front; the designs are very much focused on individual user stories; the overall system design evolves into a final design via a process of continuous refactoring (restructuring/optimising system code without changing its behaviour).
Tests	XP is regarded as a test driven methodology (TDD); Programmers and customers compile a set of tests as part of a user story document; this is done before coding.
Pair Programming	All production code is written by two programmers in a single location using a single machine.
On-site customer	A customer sits with the team full-time
Continuous integration	New code is integrated with the evolving system within a short space of time; any new code that compromises the systems' ability to pass the set of pre-defined tests is discarded.

Table 2.3 has made liberal reference to the entities, customer and programmer. In order to establish a bit of convention and provide some clarity with regards to the use of terminology pertaining to XP methodology, Lindstrom and Jeffries (2004) explained that the main role players are the customer and the programmer. The customer is a business representative who provides details regarding the system's requirements and the expected business value. Around these requirements and business value specifications, a set of test cases are developed to enable the system delivers the expected functionality. The programmer is a member of the technical team assembled to implement the customer's requirements and develop the software system. The major contributions of XP are centered on shorter development cycles, the recommendation of using an evolutionary design approach (as opposed to the 'big design up front' used in traditional methodologies), an emphasis on continuous testing and integration, the invocation of a pair programming strategy and the

requirement of having an on-site customer. The essence of XP methodology is simplicity in terms of planning, design, programming, testing and feedback (Lindstrom & Jeffries, 2004). There is a high focus on interactivity with the customer because the customer is responsible for prescribing the acceptance tests and then evaluating the software to ascertain if it delivers the intended business value. Each iteration of an XP cycle produces a working version of the software that is evaluated by the customer. Hence, there is a high priority attached to visibility of the software thereby enhancing the prospect of customer feedback. This is different from traditional software development methodology where the customer involvement is restricted to specific phases of development and the actual system is delivered completely at the end of the development cycle.

While the XP methodology has received many endorsements, Beck (1999) did concede that it was not "…a finished, polished idea" (p. 77) and it is ideally suited to small, medium sized systems where requirements are not concise and were likely to change during the course of development.

Empirical Deliberations Involving XP Methodology

An objective indictment on the effectiveness/success of XP was not easy to acquire because many of the reports in this regard have been based on anecdotal evidence (Abrahamsson & Koskela, 2004; Layman *et al.*, 2004). According to Abrahamsson and Koskela (2004) and Williams *et al.* (2004), this situation was not entirely unexpected and while empirical evidence is valued, much of the decision making regarding software development within the practitioner community did not have empirical justification. This observation resonates with the assertion by Glazer *et al.* (2008) that although it was clear, to many businesses and software engineers, that the XP attributes that prioritised rigorous customer interactions and frequent delivery of software enabled the production of superior software, this claim was not based on accurate empirical evidence. This situation 'opened' up the methodology to criticism for lacking in its ability to deliver quality software systems.

In an effort to address this situation, Abrahamsson and Koskela (2004) conducted a controlled case study on XP in a practical setting that entailed development of a system for managing research data. The development team consisted of 4 developers and the user base was in excess of 300 users. The large user base meant that there would be varying expectations of the system and as such, the developers were provided with an incomplete set of user requirements that would be modified on the basis of continual user interaction with the system. The overriding objective of the study was to set a benchmark for the performance of XP on the basis of empirical data that provided an indication of the success of core aspects of XP methodology. This would serve as a point of reference that researchers and practitioners could use in their analysis of XP methodology both holistically and also with respect to the specific practices that underpin the methodology. This strategy resonates with the suggestion by Erickson *et al.* (2005) that the main XP methods have to be studied separately to determine whether each of these methods achieve the expected levels of success.

XP and the On-site Customer

One of the significant findings of the Abrahamsson and Koskela (2004) study was that direct customer did not play a significant role in the success of the system. This outcome is commensurate with the results of a similar study by Rumpe and Schröder (2014). The suggestion that the presence of an on-site customer is not pivotal to the success of XP is contrary to the dictates of methodology as suggested in Beck (1999). However, on closer analysis, Abrahamsson and Kosokela do concede that the development team appreciated the convenience of having an on-site customer for quick system reviews, the development of user acceptance tests as well as to provide a tokenistic presence to instil a sense of sense of urgency and commitment to the development effort. Also, user involvement in the systems development effort is positively correlated with end user acceptance of the system (Bano & Zowghi, 2013; Kujala, 2003; Kundalram, 2013; Williams *et al.*, 2004). The Rumpe and Schroder study that entailed a survey of 45 software practitioners, was not conclusive in this regard. The majority of the survey responses indicated that the presence of an on-site customer would have been preferred, but it was mostly logistical problems that prevented the dedicated involvement of an on-site customer. There were also reports of instances where the on-site customer was not competent enough to contribute towards the writing of accurate user stories or the generation of adequate test criteria to ensure that valid tests were conducted on the evolving system. In these instances, the system development effort was delayed and lead to much frustration on the part of the developers as well as the on-site customer. Hence, the dedicated presence of an on-site customer would be ideal, provided the customer is familiar with the user story concept and also has the competency to write valid test criteria that could guide the system development effort in the right direction. This ideal scenario may not be always feasible, thereby compromising the integrity of the methodology. However, a compromise situation that entails sufficient involvement of the on-site customer to provide only overview detail with regards to the compilation of user stories and the generation of test conditions seems to be the most plausible resolution to the dilemma of on-site customer involvement.

XP and Design, Testing and Code Refactoring

As indicated in Table 2.3, XP is regarded as test driven development (TDD) where each feature of the system is coupled with a series of predefined tests that are compiled by the programmers and the on-site customer. It is reported in Causevic *et al.* (2011) and Layman *et al.* (2004) that TDD consists of an iterative cycle of test, development and refactoring of code with the objective of ensuring that all test cases are passed. The significant aspect of TDD is that it minimises the need for a comprehensive system design phase. The system design evolves on the basis of interaction with the on-site customer. This feature of XP may be deemed to be rather controversial within in the annals of conventional systems analysis and design literature. Historically, a comprehensive system design phase, referred to as the big design up front (BDUF), has always been part of the systems development process. However, the XP methodology with its TDD approach adopts a minimalist approach to design, thereby rendering the methodology to be in conflict with the BDUF strategy.

BDUF vs TDD

The dilemma arising out of XP's deviation from the BDUF convention is whether the lack of a comprehensive system design in favour of the iterative TDD approach will compromise system quality. In order to shed some light on this dilemma, Layman *et al.* (2004) conducted a case study at IBM involving the development of device driver software. The development of the device driver software was done using the conventional BDUF approach and at a later stage, an updated version of the software was developed using TDD. In both instances, UML was used to create designs of the system. However, while a comprehensive design model was developed for the older system, the newer system had a scaled down design model that was accompanied by a set of predefined acceptance tests. A comparative analysis on the number of defects that were identified in the code for both the older and newer versions of the device driver software was conducted and a significant outcome was that the newer system using the TDD approach had 40% fewer code defects reported. While this outcome seems to suggest that TDD is a superior methodology, Laymen *et al.* do acknowledge that the limitations of the case study approach (such as the lack of external validity and the inability to produce statistically significant results) may justify an element of caution when making generalisations on the basis of such a study.

However, a case study can provide valuable insights into the adoption and effectiveness of new technology or practice (Layman *et al.*, 2004). The insight into the benefits of TDD alluded to in the Laymen *et al.* study is however reinforced in a systematic literature review of empirical studies on TDD undertaken by Causevic *et al.* (2011). The review identified and provided a summative report on the outcome of 48 empirical studies where TDD was the main focus. The most significant outcome of this review was that TDD had a significant positive effect on code quality. This conclusion was based on reports of a lowering in the code defect density[11] once TDD was used. Also, many of the studies reported on the positive perception of software practitioners towards TDD.

[11] A software defect is a generic term for a fault, failure or error in a software product (Schach, 2008, p. 50)

Whilst the apparent benefits of TDD seems to suggest an improved software process, Causevic *et al.* warn about the limiting factors that may have a moderating effect on the claimed benefits of using TDD. A significant moderating factor that will hinder widespread adoption of TDD is the lack of experience or knowledge in the use of the methodology. This lack of expertise will have a negative impact on the code quality as well as contribute towards a less than optimal return with regards to time and budgetary constraints. Another constraint is the reports of unscheduled increases in the development time (confirmed in a previous study by George and Williams (2004)). This is attributed to the time incurred to implement a set of requirements, attempt to ensure that the acceptance tests are met and engage in code refactoring so that there is an improvement in the code quality.

The code refactoring activity, an intrinsic part of TDD, may also introduce regression faults that make it necessary to repeat all of the acceptance tests subsequent to any change in the code base, thereby increasing the development time. Depending on the organisational context, development time may be regarded as a critical factor in enabling business value (Causevic *et al.*, 2011). If a project is not completed within a given time, then it impacts negatively on the business value (Alsultanny & Wohaishi, 2009), thereby compromising the viability of the project and the methodology used to develop the project. In a study by Kim *et al.* (2012) the issue of code refactoring was examined in a case study of the Windows operating system at Microsoft. The study entailed a quantitative component where 1290 software engineers at Microsoft were surveyed as well as a qualitative component that entailed semi-structured interviews with 6 engineers who were assigned the task of refactoring the Windows 7 operating system. In the quantitative part of the study, developers were asked to critically analyse the concept of code refactoring. The reported benefits of code refactoring were improved readability of the code, improved maintainability, a lower defect rate and better extensibility of the system. The reported risks associated with code refactoring entailed the generation of regression faults and the time taken to conduct code refactoring. The significant outcome from the interviews was that code refactoring provided an opportunity to add business value to the system. In the Windows case, this was done by customising the code to make it compatible with different execution environments. From an overview perspective, it may be concluded that the time overhead incurred by these code refactoring efforts may well be mitigated by the increased maintainability that is incorporated into the system, thereby saving on additional development costs. Hence, based on the evidence presented, the XP philosophy of intensive refactoring throughout the project (Kim *et al.*, 2012) may be pivotal in improving software quality at a cost that may be repaid by virtue of a reduced maintenance overhead (which according to Schach (2008, p. 13) consumes approximately 75% of the cost of software development).

While the lack of experience in the use of TDD and the time overhead have been flagged as criteria that may impede the widespread adoption of XP, Causevic *et al.* (2011) also examined the strategy of adopting a minimalist approach towards an upfront system design. This strategy, which heralds a significant departure from traditional software development ideology, has also received much attention in Breivold *et al.* (2010) and Mishra and Mishra (2011). In each of these studies, it is reported that the lack of a BDUF approach is not seen as a hindrance or a limiting factor in the quest to develop quality software. However, Causevic *et al.* do caution that there are studies where the lack of a comprehensive design phase particularly for larger, complex systems has had a negative impact on the quality of the system. Hence, there is no definitive indictment on whether the lack of a comprehensive upfront design is beneficial or detrimental to the quality of a software system. Breivold *et al.* (2010) is of the opinion that this aspect of agile methodology should be the focus of further empirical inquisition. In a subsequent study, McHugh *et al.* (2012) conducted a survey of 20 medical device software organisations. Fifteen of these organisations had opted for a plan driven, prescriptive software process model where there is a large emphasis on upfront planning and design. It is claimed that such an approach provides the stability and point of reference for a software project that serves a 'mission critical' purpose. This sentiment is endorsed by Meyer (2014, p. 13) who is of the opinion that the agile-like stance of rejecting extensive upfront planning and design is "irresponsible" and does not auger well for the sustainability of agile methodology.

The User Story as a Proxy for BDUF

Whilst this opinion resonates well with the dictates of traditional software engineering practice, the McHugh *et al.* study did reveal that the software practitioners were of the opinion that user stories are an adequate form of upfront planning and provide the necessary stability that a distinct design phase would provide for the traditional methodologies such as the Waterfall approach. These observations give rise to a paradoxical situation where the academic fraternity is wary of diminishing the relevance of a comprehensive upfront design effort, while the practitioner community is gravitating towards a strategy that entails diminishing of the overheads that would be incurred if too much time and effort is spent on the analysis and design phase of the development lifecycle.

As a concluding observation regarding the design issue, the lack of a comprehensive upfront design effort may not necessarily hinder the software process. In some instances, where the system requirements may be deemed to be volatile, the XP methodology consisting of user stories, TDD and code refactoring may be ideal. However, in other instances where the system is deemed to be complex or it serves a 'mission critical' purpose, the more prescriptive methodologies such as the Waterfall model with a BDUF focus will be preferred.

XP Methodology and Pair Programming

Another aspect of contention regarding XP is the programmer-centric nature of the methodology. XP is not reliant on expert contributions in the areas of systems analysis and design (Crawford *et al.*, 2013). Most of the analysis, design and coding is done by two programmers who work together on the same programming task using one computer and one keyboard (Dick & Zarnett, 2002; Hannay *et al.*, 2010). Programmers work together in pairs and develop simple designs that represent a high-level abstraction of the system. XP methodology entails the development of code using pair programming as well as rigorous testing of the code until it conforms to a set of acceptance tests that have been specified upfront, in collaboration with the business stakeholder. While the minimalist design approach has been deliberated upon in the previous paragraph, the invocation of a pair programming strategy to develop code for the system has received much focus in software engineering academic literature (Abrahamsson & Koskela, 2004). Pair programming has been researched extensively from an academic and a practitioner oriented context. In the academic setting, it has been claimed that pair programming enhances code quality and programmer confidence in the software development process (Slaten *et al.*, 2005) and is also pivotal in reducing the number of errors that may be found in a code review (Tomayko, 2002). These claims have been corroborated in empirical studies by Radermacher and Walia (2011), Wood *et al.* (2013) and Abeyratne (2014). The strategy used in these studies was to make use of a control group that entailed a comparison of the quality of programming solutions when using individual programming[12] in contrast with pair programming techniques. In all of these studies, it was apparent that the pair programming strategy yielded higher quality software solutions than the individual programming strategy. In terms of students' perceptions of both programming strategies, there was a strong preference for pair programming as opposed to individual programming because of better defect identification and code quality (Tomayko, 2002), greater synergy and problem solving ability (Abeyratne, 2014) and the ability to deliver a solution in a shorter time duration (Radermacher & Walia, 2011). In the latter study, all of the subjects who used pair programming indicated a preference for pair programming over individual programming.

The discussion in the preceding paragraph attest to the benefits of using a pair programming strategy in an academic setting. However, all the authors whose studies formed the basis of that discussion expressed an element of caution with regards to extrapolation of the conclusions made to any other context. The implementation of a control group based experimental stance would be difficult to achieve in an industrial setting because of the unpredictable environmental factors that will have a confounding influence on the outcome of such a study (di Bella *et al.*, 2013). Also, any sort of software engineering research efforts in an industrial

[12] Individual programming is a reference to the traditional approach to write computer programs where a single developer works alone on a development task (Gallis *et al.*, 2003)

setting is generally difficult to achieve. This assertion is based on the suggestion by Gallis *et al.* (2003) that software engineering research is complex and involves a mix of technological elements with the human and organisational influences. In most studies on the effectiveness of pair programming, the possible moderating effect that social and organisational issues may have had on the outcome of these studies have largely been ignored (Arisholm *et al.*, 2007). Although Gallis *et al.* (2003) did provide a framework for evaluating the effectiveness of pair programming, researchers have opted for issuing disclaimers that allude to the possible moderating influence of the social and organisational issues on the outcome of these studies.

In an attempt to remedy this deficiency of knowledge regarding pair programming, di Bella *et al.* (2013) conducted a 14 month case study to investigate the effectiveness of pair programming in an industrial setting. The analysis of the results of this study seems to suggest that pair programming contributes to a small reduction in the defect density of code used for the implementation of user stories. However, the use of pair programming was pivotal in ensuring that no new defects were introduced when efforts were made to rectify the original defects.

The outcome of this inquisition on the benefits of pair programming may not be conclusive from a scientific perspective. However, the empirical and anecdotal evidence seems to suggest that pair programming makes a significant contribution towards enhancing code quality by enabling a reduction in the defect density found in code artefacts during the initial stages of development and more significantly, during the maintenance phase. There is also greater synergy that prevails between paired teams and within the teams themselves. While this synergy is beneficial to the overall morale of the development team there have been suggestions that pair programming does not necessarily translate to better productivity with respect to delivery of functionality by the development team.

A Review of XP Methodology

The expansive coverage of issues underpinning XP methodology from a historical perspective is deemed pertinent as part of the current discourse because it provides coverage of specific aspects of contention with regards to methods that are intrinsic to XP and the broad framework of agile methodology. In an opinion piece on lightweight software development methodologies with a specific focus on XP methodology, McCormick (2001) alludes to the importance of appreciating the essence of XP methodology within specific contexts. A cautionary note is issued that XP and agile methodology in general should not be seen as a 'prescriptive panacea' to the software crisis. However, the agile approach is reflective of a stance to software development that is adaptive to the technology being used, the size and complexity of the software project, the criticality of the system being developed as well as the regulatory and cultural constraints imposed on the systems development effort. Given all of the afore-mentioned parameters that influence the software development process, the 'sweet spot' area for XP has been identified as software systems that are perceived to be smaller and serves a less critical purpose (Beck, 1999; McCormick, 2001). Although this is a delimiting indictment on the applicability of XP methodology, McCormack asserts that "…XP ought to be one of the tools in our bag of tricks" (p. 110) and serves a niche area of software development.

The seemingly capricious state of XP and agile methodology in general prompted Dybâ and Dingsoyr (2008) to conduct a systematic review of published literature on agile methodology. The review, that included only research papers based on empirical evidence, acquired seminal status by providing a definitive, overview knowledge of XP and agile methodology. The study entailed a meta-analysis of 36 empirical studies on agile software development (narrowed down from 1996 articles). Seventy-six percent of these studies were based on XP methodology. While the Dyba and Dingsoyr review was one of the most comprehensive regarding XP and general agile practice, the results of the review were defended by disclaimers that the analysis was not definitive and were based on the evidence of a limited number of empirical studies that had been conducted up until 2005. This disclaimer was coupled with a call for further empirical studies on the viability of using XP and agile methodology in general. The outcome of the review shows an overriding positive indictment on XP methodology, although a few serious limitations were noted. The limitations included aspects such as:

- The presence of the on-site customer is not sustainable over a long period of time. This assertion is commensurate with a similar sentiment expressed in Martin *et al.* (2004) as well as Rumpe and Schröder (2014) where it is claimed that it is challenging for the onsite customer to become accustomed to the prevailing organisational culture as well as conform to the social dynamics of the interaction with the development team over a prolonged period of time. The relationship between the on-site customer and the development team has a potential to degenerate into an adversarial one because of an arrangement whereby a non-technical stakeholder is accorded peer status to a technically oriented software engineering team. Another issue that plagues this aspect of XP methodology is the logistics of adjusting to the potential disorganisation that may result as a consequence of the prolonged absence of the on-site customer from the regular role played within the organization;
- It is difficult to introduce agile methods and XP in particular into large and complex projects;
- In some instances, pair programming proved to be frustrating and many projects ended with the development team resorting to individual programming for the latter stages of the project. Although this assertion was more tentative than definitive, there were reports that pair programming was beneficial only in the initial development phases. Hereafter, there was a tendency for programmers to resort to individual programming in the latter phases of development or during the maintenance phase;
- There were concerns regarding the lack of analysis and design at the initial stages of development. This lack of a BDUF strategy resulted in a development philosophy that was unstructured and ad hoc and not ideal for complex systems;

- XP and agile methodology in general is not ideal for large projects.

While these limitations provided a cautionary perspective, the majority of the studies in the review issued a positive indictment on XP methodology from the following perspectives:

- XP methodology is a huge improvement from the Waterfall methodology;
- XP methodology reduces the development time as well as the cost overruns that have become intrinsic to the software development process;
- XP methodology enhances productivity of the development team and customer satisfaction with the final product;
- Pair programming was positively received by XP developers and there is empirical evidence to suggest that the pair programming strategy leads to higher quality code.

The factors listed are significant in upholding the perception that XP methodology has received widespread acceptance in the software development community. However, it is in the social and organisational realms where the major benefit of using XP methodology has been reported (Dybâ & Dingsoyr, 2008). A study by Sharp and Robinson (2005) that involved 3 companies of varying organisational and physical infrastructure reported that XP was sufficiently 'malleable' to be successfully implemented in all 3 companies. Based on the results of an empirical study, Mannaro *et al.* (2004) claimed that the adoption of XP methodology lead to greater productivity and job satisfaction. Tripp and Riemenschneider (2014) did however caution that claims such as these were generally not based on any theoretical underpinning, deemed to be pivotal for operationalising abstract concepts such as job satisfaction. In order to remedy the situation, Tripp and Riemenschneider used the Job Characteristic Model (JCM), claimed to be one of the most tested theoretical models in social science, as a theoretical basis develop a questionnaire that was disseminated to a sample of 104 software development practitioners in the United States. The focus of the questionnaire was to establish a possible relationship between the implementation of agile methodology and job satisfaction of the practitioners. Whilst this study did not focus on any specific agile method, many of the XP techniques such as test driven development, code refactoring and pair programming formed the basis of the analysis that was performed.

The Success of Agile

The results of the analysis indicated that there is a significant positive relationship between the implementation of agile methods and job satisfaction. While the Tripp and Riemenschneider study may have been obfuscated by general principles of agility and not just XP methodology, the benefits of XP methodology from an organisational and social perspective are confirmed in Tessem and Maurer (2007), who used a single case study specifically focused on XP, and de O Melo *et al.* (2012) who used a multiple case study approach that entailed a mix of agile methods, one of which was XP. A periphery observation from the O Melo *et al.* study was that while XP was given some focus, the dominant agile methodology of development was Scrum. From a historical perspective, the advent of agile software development methodology in the late 1990s was accompanied with a specific preference for XP because of the vast amount of literature that was available on the methodology (Fowler, 2005). Also, the test driven development (TDD) culture as well as the focus on the engineering aspects of software development made XP an attractive option for practitioners who were venturing into the realm of agile software development for the first time. However, over a period time, the demand for a methodology that focused on the management aspects of the software process rather than the engineering or the technically oriented coding aspects, resulted in a migration within the agile domain, from XP methodology to Scrum methodology.

2.4.4 The Migration from XP Methodology to Scrum Methodology

According to Fowler (2005) Scrum attaches importance to the management aspects of software development, dividing development into concise time periods referred to as iterations or sprints. The intention is to establish closer monitoring and control of the software process by engaging in daily meetings referred to as the daily scrum. As was observed in the multiple case study by de O Melo *et al.* (2012), a viable arrangement would be to combine Scrum and XP thereby enabling the attainment of a project management and software engineering focus at the same time. This complementary coupling has been pivotal in establishing Scrum and XP as the 2 most popular agile methodologies (Fitzgerald *et al.*, 2006). Theoretically, the project management value provided by Scrum and the software engineering value provided by XP would render an integration of these methodologies an optimal one. This strategy of selecting only parts of agile methods that are perceived to be 'fit for purpose' within an organisational context is referred to as method engineering (Henderson-Sellers, 2006) or a meta-method approach (Fitzgerald *et al.*, 2006) where a methodology is developed from existing method fragments.

In order to establish the viability of the method engineering approach, Fitzgerald *et al.* (2006) adopted a case study approach to investigate the customisation of XP and Scrum at the software engineering division of an international organisation. The study involved an interpretive, exploratory strategy that entailed a series of interviews with software engineers who were responsible for the customisation of XP and Scrum. The interviews were designed to obtain feedback from the software engineers on the basis of their continuous monitoring and evaluation regarding this customisation process over a 3-year period. The culmination of the study was a presentation of a tentative framework for software development that consisted of a hybrid arrangement of XP and Scrum based methods, illustrated in Figure 2.10

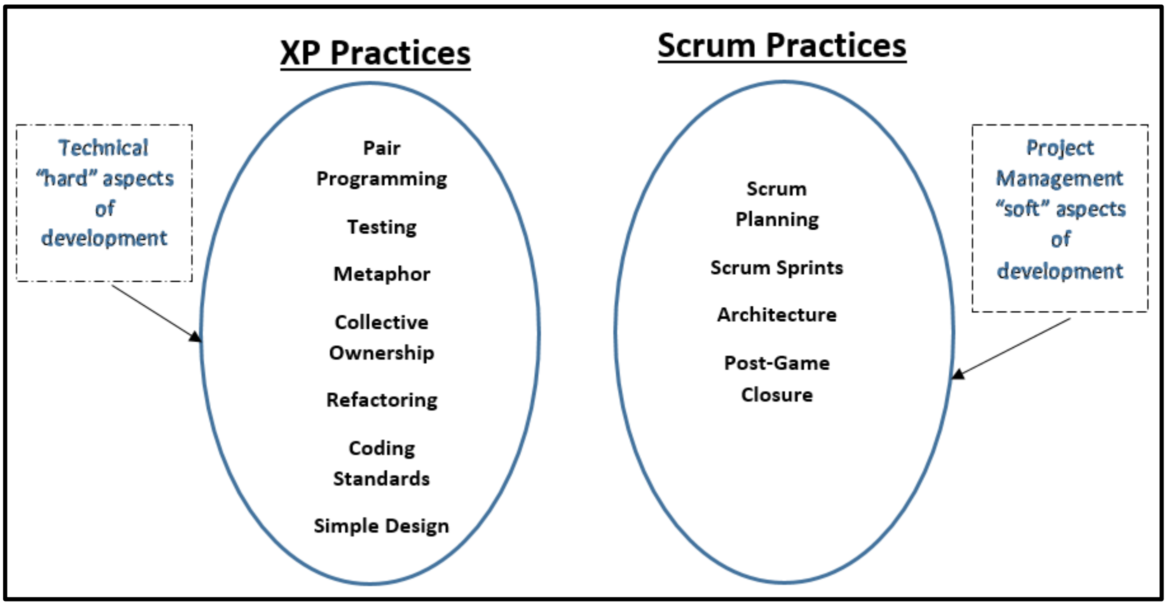


Figure 2.10: An Integration of XP and Scrum Methodology (taken from Fitzgerald *et al.* (2006))

The framework proposed by Fitzgerald *et al.* (2006) consists of an invocation of selected methods from XP and Scrum where the objective is to leverage the benefits of the methods in a complementary manner so that the customisation process resonates with the prevailing socio-technical environment in an organisation. As illustrated in Figure 2.10, the specific context of the problem domain necessitated the inclusion of only a subset of the full suite of XP methods. The omission of certain XP practices such as the presence of an on-site customer, continuous integration of code artefacts and a 40-hour working week was necessitated by the requirements of the project. Also, the practice of pair programming and the use of a metaphor to describe the main system functionality was used sparingly because it was not practically feasible to enforce either practice.

The Scrum contribution is centered on the value provided by the sprints which are used as a node of the development process around which much of the planning and control is invoked. This approach obviated the need for the 'planning game' approach advocated in XP. In this way, the customisation process entailed the selection of specific methods from a palette of XP and Scrum methods so that it became a 'best fit' for the situation at hand. From a holistic perspective, the complementarity of these methods is manifested in the support provided by XP for the technical aspects of the systems development effort while the project planning and tracking support is provided by Scrum. It should be noted that Paulk (2001) warns against the adoption of a fragmented, method based customisation of an agile methodology. However, the successful customisation of agile methodology where the agile methods are chosen in an a la carte manner as suggested by Fitzgerald *et al.* (2006) is corroborated by Treacy *et al.* (2008) who used a multiple case study approach involving 3 companies that used customised versions of XP as well as an integration of XP and Scrum methodologies. The findings of this study concur with the claims by Fitzgerald *et al.* (2006) that XP and Scrum can be modified to match the requirements of the project as well as organisational and management norms and expectations.

The preceding discussion alludes to the positive reports regarding the complementarity between XP and Scrum. While this integration of XP and Scrum has been endorsed by the empirical findings of Fitzgerald *et al.* and Treacy *et al.*, both these studies are subjected to the limitation that XP and Scrum have not been assessed to ascertain whether they may each be individually superior to the customised versions. Fitzgerald *et al.* do concede that the problem with agile method engineering is that only those parts of an agile method that are perceived to be 'fit for purpose' are included in the customised version, thereby running the risk of compromising the holistic benefits that may be achieved when using the methodology in its original format. These concerns regarding the method engineering of agile methods coupled with the observation that there was a need for better project management guidance in the adopted methodology resulted a gradual decline in the popularity of XP as well as the XP and Scrum combination. This trend is explored in the succeeding discussion.

According to Dybâ and Dingsoyr (2008), many of the academic studies published until 2005 focused on XP. During this period, the prominence of XP in the academic community was also aligned to its prominence in the practitioner domain, as illustrated in Ambler's survey (reported in VersonOne, 2011) of 4232 software practitioners who were using agile methodology. A significant outcome of this study was that the highest adoption rate of all the agile methodologies used, was XP (57%). Vijayasarathy and Turk (2008) reported on the results of a similar study as the one conducted by Ambler. This study was underpinned with a theoretical framework that consisted of Rogers' Diffusion of Innovation Theory(DOI) (explained in Rogers, 1983). One of the objectives of the study was to survey a sample of early adopters of agile methodology and to establish which agile methods were most popular. Using Rogers' DOI theory as a theoretical base, the study examined the preferences of early adopters of agile methodology and concluded that there was a strong preference for agile methods that were intrinsic to XP methodology. This included TDD, pair programming and code refactoring. The early adopters of agile methodology had a preference for the engineering oriented aspects of the methodology (as embodied by XP) rather than the management aspects of software development (as embodied by Scrum). However, the 'later adopters' of agile methodology had a preference for the methods that enabled better control and management of the software development process. This trend that represented a need for better control and management of the software process rather than focusing on the operational aspects of the process, manifested in Scrum replacing XP as the more popular agile methodology.

From a practitioner perspective, the results from Ambler's survey were slightly different to the global survey conducted by VersionOne[13] (see VersionOne, 2013) in 2013. The results from the 2013 survey reported Scrum as the most adopted agile methodology (40%), followed by XP (23%) and then a hybrid methodology based on Scrum and XP (14%). The results suggested a trend that Scrum was regarded as the most popular methodology, followed by the Scrum hybrids and then XP. This trend continued in an unabated manner and Scrum has become established as the most highly adopted agile methodology. The preceding assertion resonates with the prediction by Dybâ and Dingsoyr (2008) that Scrum methodology was gaining substantial focus and would soon replace XP as the most popular agile methodology. In the 2015 VersionOne survey, a total of 3880 software

[13] VersionOne is a software organisation that conducts global survey on issues pertaining to ASDM in order to enhance the success of the agile management tools that the organisation develops.

practitioners were surveyed on their use of agile methodology. The results indicated that 58% of the respondents were using Scrum as the default methodology for software development. The Scrum/XP hybrid was the 2nd most preferred methodology with exclusive use of XP reported at less than 1%.

The Migration from XP to Scrum

A significant outcome of the 2015 survey was an affirmation of the trend regarding the decreasing reliance on exclusive use of XP methodology and an increase in the adoption of Scrum methodology. The popularity of the Scrum/XP hybrid was however an indication that many agile concepts attributed to XP methodology (such as TDD, code refactoring, pair programming and continuous code integration) were perceived by practitioners as being quite useful to the software development process. The significant preference for a Scrum based approach was quite distinctive in this regard. A possible explanation for this trend is that the regular 'stand up' meetings intrinsic to Scrum enables more frequent reporting on the status of a project thereby facilitating better project management and ultimately ensuring accountability on the consumption of project resources. This overwhelming preference for Scrum warrants a closer inspection of the underlying techniques used in the methodology.

2.4.5 Scrum Methodology

As was the case with XP, Scrum methodology was proposed as an alternative to the sequential, 'heavyweight' methodologies that were not flexible enough to accommodate the changes in user requirements as well as the volatile environment in which the software systems were required to operate. The origins of the Scrum methodology can be traced back to a paper titled "The New Product Development Game" by Takeuchi and Nonaka (1986) where they proposed a more dynamic approach to development of new products. According to Takeuchi and Nonaka, the parameters governing new product development have changed and the development process has to incorporate speed and flexibility as pivotal factors in the criteria used to determine successful product development. The old approach of product development where each phase of development was specified completely with great precision and detail before a new phase could commence compromised the parameters of speed and flexibility. Schwaber (1997) explains that the Scrum development methodology enables speed and flexibility by incorporating strategies that enable the quick delivery of a working version of the system based on a broad set of system specifications. The development team acquires good working knowledge of the main objectives of the system and then engages the Scrum methodology to achieve the main objectives of the system. The strategy is analogous to the game of rugby (the analogy is explained in greater detail in the article by Takeuchi and Nonaka (1986)) where the ball gets passed within the team as the team moves as a unit up the field until it reaches the opposition's goal line. The idea here is that the team is provided with an overall objective and the team members are given the autonomy of deciding on the strategy of how to best achieve the objective in the quickest possible time. Although project teams are given the autonomy of deciding on the strategy that will enable optimal progress, management have the opportunity of establishing 'checkpoints' to prevent instability and avoid the impending chaos of an uncontrolled process.

The Scrum methodology was adapted for the software industry by Jeff Sutherland and jointly presented at the Object-Oriented Programming, Systems, Languages & Applications '95 Conference (OOPSLA '95) with Ken Schwaber, who formalised Scrum methodology in a paper titled "Scrum Development Process" (see Schwaber (1997)). According to Schwaber, Scrum is a methodology that is an enhancement of the iterative and incremental software process model (SPM) and an improvement of the Spiral SPM which he claims is a linearly oriented model where each phase of development precluded activities that belonged to any other phase of development. This lack of flexibility is offset by Scrum methodology that enabled software development teams to operate with maximum flexibility that enhances the prospect of producing "orderly systems under chaotic circumstances" (p. 8). The main difference between the Waterfall, Spiral and Iterative and Incremental SPM's and Scrum is that the core component of the systems development activity, referred to as a Sprint (illustrated in Figure 2.11). The inner workings of a Sprint are assumed to be completely unpredictable, prompting

Takeuchi and Nonaka (1986) to accord 'black box' status to a Sprint. The skeletal structure of Scrum methodology as illustrated in Figure 2.11, makes reference to a project initiation phase, a development phase referred to as a Sprint and a project closure phase that is planned and prescribed.

The Sprint is the Central Construct of Scrum

The Sprint phase, which forms the central construct of Scrum methodology is subjected to the complexities of a volatile development and target environment as explained in Takeuchi and Nonaka (1986). These complexities render the Sprint phase as highly unpredictable. From a holistic perspective, the strategy used to mitigate the risk inherent in the unpredictability of a Sprint phase is to implement an iterative approach as illustrated in Figure 2.11. The iterative demeanour, embodied by the Sprint phase, provides an opportunity to impart controls that enable ongoing assessment of the systems development effort so that risks are identified and mitigated. This risk driven approach does not seem to be radically different from previous iterative SPM's such as Boehm's risk driven Spiral process model. This observation is endorsed by Bannerman *et al.* (2012) who commented that Scrum methodology is an expedited version of the Spiral SPM with a few minor differences. The Spiral SPM is classified as an iterative model that is managed by a risk assessment exercise at the end of each phase of development thereby ensuring an accountable process. However, development proceeded slowly because of the inability of the Spiral model to impart a sense of urgency and flexibility to the software process.

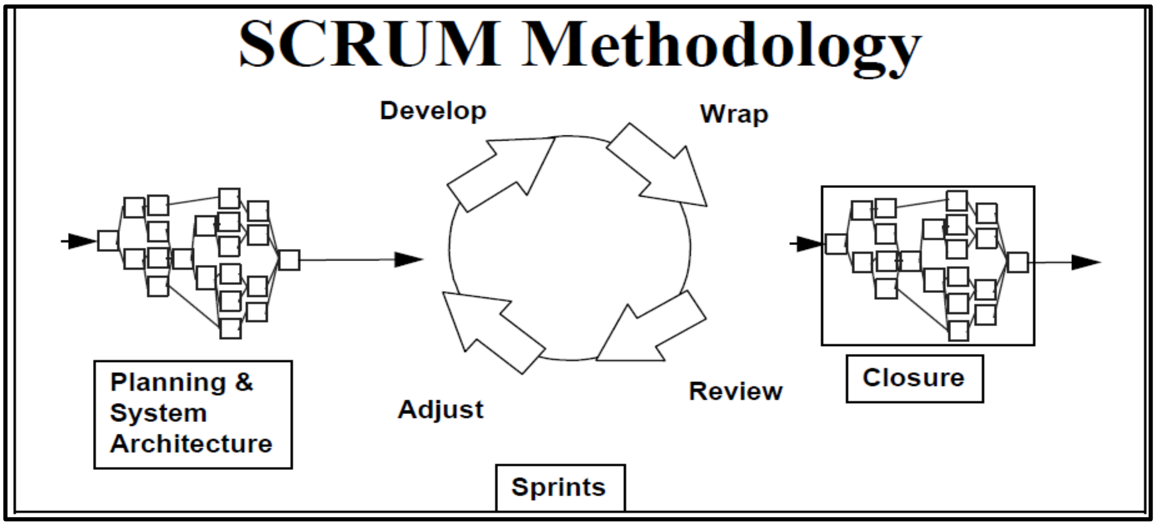


Figure 2.11: An Overview of Scrum Methodology by Schwaber (1997)

These shortcomings of the Spiral SPM are mitigated by Scrum techniques such as Sprints, time-boxing, maintenance of a product backlog and the holding of daily Scrum meetings. As can be observed in Figure 2.11, there is still a prescribed phase for planning (development of the product backlog (PB) list; prioritising system features for release based on the level of importance; risk assessment and acquisition of management support and approval; analysis and lightweight design of the planned system release) and system closure (invocation of integration and system testing, concluding of system documentation and training artefacts to facilitate smooth integration of the newly developed system into the organisational infrastructure). The core phase of the Scrum methodology is the Sprint which is a time-boxed iteration of the development cycle. The time restriction placed on the development cycle imparts a sense of urgency and the daily Scrum meetings that occur within the Sprint facilitate a sense of accountability in terms of the quality of the evolving system and the resources consumed. The illustration in Figure 2.11 provides an overview of the methodology. Greater detail of the Scrum methodology is provided by Sutherland *et al.* (2012) and illustrated in Figure 2.12.

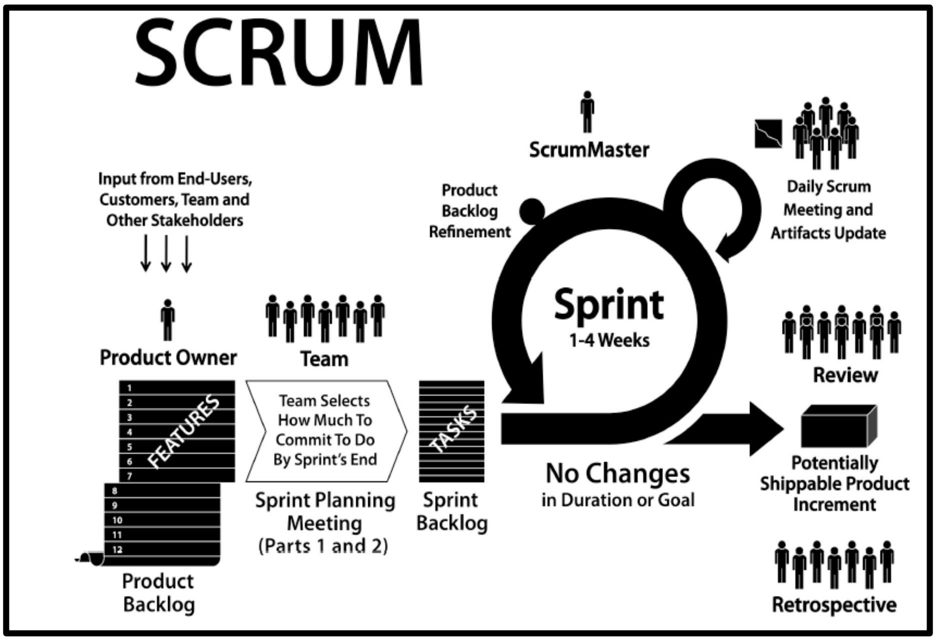

SCRUM
Input from End-Users, Customers, Team and Other Stakeholders
Product Owner
Team
ScrumMaster
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Review
FEATURES
1
2
3
4
5
6
7
8
9
10
11
12
Product Backlog
Team Selects How Much To Commit To Do By Sprint's End
Sprint Planning Meeting (Parts 1 and 2)
TASKS
Sprint Backlog
No Changes
in Duration or Goal
Potentially Shippable Product Increment
Retrospective

Figure 2.12: Scrum Methodology (taken from Sutherland *et al.* (2012)

As can be ascertained from the representation of Scrum methodology in Figure 2.12, there is a significant emphasis on ensuring a high level of interactivity between the system's stakeholders in order to uphold the primary objective of developing software that is made accessible for users without incurring the delays of exhaustive planning and design. An important aspect of Scrum is that there is only high level focus on analysis and design modelling or on documentation. The inner details or detailed specifications of analysis and design modelling is left to the discretion of the development team and tends to evolve with the system's coding phase that occurs during the Sprints. While this philosophy is congruent with agile methodology in general, it is the inner workings of Scrum that differentiate the methodology from other agile methods. The inner workings are centered on a Product Backlog (PB) and iterative work cycles named Sprints (as illustrated in Figure 2.11 and Figure 2.12).

The PB contains a prioritised list of all items relevant to a specific product. The Product Owner is the person responsible for managing and controlling the PB. One of these responsibilities entails the setting of priorities for each item in the PB. The PB is continually updated to reflect changing customer requirements as well as changes in the development and business domain. The PB serves the purpose of providing the customer with an opportunity to refine or enhance system requirements to reflect changes in the business domain, thereby ensuring that Scrum maintains a demeanour of flexibility that is aligned to the principles of agility. However, as the system specification evolves to the point where the requirements have been completely defined, Scrum tends to enter a phase of intensive development that is not fully amenable to further changes in the system specification.

The Scrum Notion of Backlog Management

Those requirements from the PB that have been refined and confirmed by the client are decomposed into tangible development tasks that are recorded in the Sprint Backlog (SB) where it becomes the immediate focus of all development activities. The SB is an output of the Sprint Planning Meeting and consists of the tasks for the Sprint derived from the PB. Each Sprint has a one month time horizon together with a definition of what is to be developed, a flexible design and plan to guide the development process over the prescribed time period (Sutherland & Schwaber, 2009). The 'time-boxed' or prescribed time period is coupled with the understanding that the requirements underpinning a Sprint task will be 'frozen' for the duration of the Sprint providing the development team with the stability that is required to enable the completion of the Sprint on time. However, at the end of a Sprint the development team has an opportunity to review the functionality of the evolving system with the system stakeholders. At this juncture, changes are accommodated and risk analysis is performed to determine whether the product sustains its feasibility. From the developers' perspective, the Sprint phase provides the development team with an opportunity to conduct themselves in an autonomous manner, with little management interference.

The Sprint phase provides the development team with an opportunity to exercise their expertise, intuition and creativity to facilitate the completion of the Sprint task. The PB and SB provides management with a strategic conduit to conduct monitoring of the overall progress of the system. Development teams make use of a project management artefact named a Burndown Chart that graphs the estimated work remaining against time thereby providing a quick reference from which a decision regarding the development schedule of the system can be made. The PB is also used as a point of reference to re-negotiate system priorities and functionalities with the client as well the development team. Success of Scrum methodology hinges on the synchronisation of the 'ceremonies' (Cho, 2008) or processes that underpin the methodology.

The Role of the Scrum Master

A significant role player is the Scrum Master, who has the responsibility of ensuring that this synchronisation does take place and the values and rules of the methodology are enforced. The Scrum Master is responsible for ensuring that there is sufficient management control over the development process and there is an enabling environment for the development team to obtain optimum productivity (Cho, 2008). A Scrum Master may be viewed as the coach of the team and has the primary responsibility of coercing the team to deliver on the expectations of each Sprint cycle and ultimately to enable the delivery of the product on time, within budget and meets with the customer's expectations. However, a significant observation is that the Scrum Master does not have any managerial control over the Scrum team. According to Cohn (2006), a Scrum Master has the task of ensuring that the team members accord maximum focus to the Sprint. The Scrum Master is essentially the team facilitator who organises the logistical and the operational requirements of the development environment so that the development team is not distracted by organisational issues.

This role is technically different from a project manager (PM) and it is not the task of the Scrum Master to provide daily direction to the team or to assign team members to individual tasks as is the responsibility of a PM. The PM's role is technically subsumed collectively by the PO, the Scrum Master and the development team members. The traditional project steering role played by a PM has been completely re-defined (possibly leading to a source of confusion (Nkukwana & Terblanche, 2017)). Such changes to the traditional mode of software development as embodied by the new roles, ceremonies and artefacts of Scrum methodology is given comprehensive coverage in the definitive guide to the methodology published by the Scrum Alliance (see Sutherland & Schwaber, 2009). These architectural changes to the software process have a combined effect of ensuring that there is a convergence of expectations of the evolving software system from management, the customer and the development team so that software is produced in a flexible, highly controlled and structured manner. The coherent integration of structure and flexibility to the Scrum methodology qualifies the perception that Scrum is the panacea to the problems associated with traditional software process models that were criticised for being either too prescriptive or lacking in management control. However, these virtues of the methodology need to be subjected to scrutiny from an academic and a practitioner perspective.

A Review of Scrum Methodology

The preceding discussion provides an overview of Scrum methodology. A plausible conclusion emanating from this discussion is that Scrum is a methodology that obviates the shortcoming of traditional software development methodologies by enabling flexibility of requirements by virtue of its iterative demeanour and continual PB reviews; it also provides an opportunity to impart project management control by virtue of the Sprint and PB review meetings as well as a 'window of opportunity' within each Sprint cycle that provides the software development team with an autonomous environment thereby minimising the overhead of too much bureaucratic control. System development is centered on quick progress with regards to producing working software. The analysis and design phases are interwoven into the software coding phase thereby minimising the prospect of incorporating extravagant analysis and design phases that are document-driven.

Prioritising Working Software over Documentation

This attribute is fully aligned to the agile principle of prioritising working software over comprehensive documentation and in many instances, the code itself becomes a proxy for system documentation (Cho, 2008; Turk *et al.*, 2014). Holistically, the 'refactoring' of the traditional software process model that has seen a change in nomenclature from analysis, design, implementation, testing, deployment and documentation to the Scrum based nomenclature of Product Backlog, Sprint Cycle, Review meetings and product increment has generally been positively received by the practitioner (Mann & Maurer, 2005; Sutherland, 2001; VersionOne, 2013) and academic (Dybå & Dingsøyr, 2008; Kropp & Meier, 2015; Vijayasarathy & Turk, 2008; Vlaanderen *et al.*, 2011) communities. A significant contributor to the positive attitude towards Scrum methodology is the attribute of ensuring early visibility of the system. In a case study by Bannerman *et al.* (2012) that involved a software development project that leveraged Scrum methodology to facilitate development in a distributed development environment that entailed a single Scrum team working in the United States of America (USA) and another 4 teams working in Australia, it was the Scrum philosophy of ensuring visibility of the developing system that enabled developers to transcend the logistical challenges of working in a 'global space'. Responses from the software developers and project manager in the Bannerman *et al.* case study included comments such as:

> *…with Waterfall, you don't know if anything has been done for months; with Scrum, you know in a matter of weeks (p. 5313)*
>
> *The best thing about Scrum is project visibility…you have to deliver every two to four weeks for inclusion in a release so it is very visible if you don't (p. 5313)*

In a multiple case study consisting of 12 semi-structured interviews with Scrum practitioners in 2 organisations in South Africa, Tanner and Seymour (2014) found that practitioners had a high level of intrinsic and extrinsic motivation to implement Scrum methodology. The high levels of intrinsic motivation lead to a sense of 'enjoyment' and 'passion' in using the methodology which in turn enabled an easy and seamless transfer of knowledge of the methodology between practitioners. The daily Scrum meetings provided a platform for the knowledgeable members of the Scrum team to impart their expert knowledge of the methodology to their peers thereby enhancing their credibility in the Scrum team (a symptom of extrinsic motivation).

This endorsement of Scrum methodology from a social perspective may be referenced to the potential for the methodology to enhance collaboration between system stakeholders (during system planning and PB identification) and fellow team members (Sprint planning and daily Scrum meetings). There is also a sense of intrinsic and extrinsic motivation to leverage the dynamism inherent in the methodology to obtain optimal benefit in terms of the quality of the system that is developed. The social benefits of using Scrum methodology are usually reported via interpretivist studies, such as the Tanner and Seymour study that does not have an explicit purpose to generalise the results beyond the confines of the context from which it is reported. However, the socially oriented studies on agile methodology and Scrum in particular provide a meaningful, deeper understanding of the value that the methodology provides from a human perspective.

The acquisition of large scale empirical data on the value of using Scrum methodology from a technical perspective is not easy to achieve because organisations do not have the luxury of conducting extensive experiments or engaging in longitudinal studies to validate the success of the methodology (Mahnič, 2008). Many of the reports regarding the success of the methodology are either anecdotal, experience based or based on case studies (Li *et al.*, 2010; Serrador & Pinto, 2015). The lack of empirically based academic research on the use of agile methodology maybe somewhat alleviated by the comprehensive. industry-based surveys conducted in VersionOne (2015) and VersionOne (2016). In both these surveys conducted with a sample in excess of 3000 software practitioners, reference is directed to the global adoption rates of agile methodology. The discerning trend is the emphatic endorsement of Scrum (58%) and the Scrum/XP hybrid (10%) as the methodology of choice for software development. The main reason advocated for the popularity of Scrum are the attributes of greater product visibility, the ability to handle changing requirements and greater team productivity and team morale. While these surveys provide a broad global view of trends in agile methodology adoption, the academic literature on the subject provides a greater depth of understanding on issues pertaining to agile and Scrum methodology adoption.

Concerns Regarding Lack of Documentation

In a case study of a company that employed Scrum methodology for most of its Web based projects, Cho (2008) reported that many of the software developers were not comfortable with the Scrum strategy of minimising the effort to produce quality documentation. The lack of quality documentation compromises system quality when it comes to maintenance. The problem is exacerbated when members of the original development team are no longer accessible to provide input into the maintenance phase. The reservations regarding Scrum's lack of quality documentation is also aligned to the results from a study by Flora *et al.* (2014) who surveyed 130 software developers in order to comprehend the strengths and weaknesses of agile methodology in general. Although the study had a focus on agile development in general. Scrum was reported to be the most widely used and the lack of quality documentation was flagged as the main source of concern regarding the adoption of agile methodology.

In a 2009 survey of 1298 software professionals on the adoption of agile methodology (see West & Hammond, 2010), conducted by the Forrester Research Organisation, Scrum was reported to be the most adopted of the agile methodologies. However, a significant outcome of this survey is that there were many instances where the principles underpinning agility in general and more specifically Scrum methodology were 'tampered' with. One of these is the principle of prioritising the delivery of working software at the expense of documentation. Development teams were forced to decrease the priority attached to working software and attach greater priority to the documentation of the evolving system. The need for this intervention was to ensure that the maintenance teams were provided with an enabling environment to handle modifications to the system without incurring too much overhead to establish the logic used during the development of the system.

Another criticism of Scrum methodology and agile methodology in general is the issue of scalability. It is reported in Lindvall *et al.* (2002) that the lack of documentation compromises the ability of an agile methodology to scale-up to bigger development projects that involves more than a single development team.

The Scrum of Scrums

However, Sutherland (2001) had a differing view based on the success he achieved in using a tactic named the Scrum of Scrums for the development of a large scale system for a healthcare software system where the development effort consisted of more than a hundred programmers. Sutherland used this experience to make the claim that Scrum scales up quite well. The Scrum of Scrums strategy basically entails adding on further layers of Scrum meetings depending on the size of the development effort. In the Sutherland case, there was the routine daily frontline Scrum meetings, a series of weekly Scrum of Scrum meetings that consisted only of representative members from the individual Scrum teams as well as a monthly Scrum meeting consisting only of business managers. While the implementation of the Scrum of Scrums strategy worked well in the Sutherland case, it must be noted that this an organisation-wide development effort that required full commitment by all stakeholders including company executives. This is an idealistic setting that worked well in an organisation where software development is the main business. However, this is not always the default situation and quite often, the development effort is fragmented and involves developers who may be working on other projects or physically located in a dispersed setting (Turk *et al.*, 2014). This situation is not commensurate with the demands of Scrum methodology where one of the underlying assumptions is that development should be conducted by small (5 to 10 members), co-located teams where there is significant reliance on face-to-face communication (Rising & Janoff, 2000; Turk *et al.*, 2014; Vijayasarathy & Turk, 2008).

According to Turk *et al.* the strategy of using the Scrum of Scrums approach to mitigate the challenge of not having co-located teams especially for a development effort that involves more than one team may not necessarily achieve the desired level of success. They go on to make the claim that in such situations, less agile methods such as comprehensive documentation, change control and better upfront system design are more applicable for large teams. From an empirical perspective, Paasivaara *et al.* (2012) conducted a multiple case study to understand how organisations managed the co-ordination across multiple Scrum teams working in a distributed environment. The data collection effort entailed 58 semi-structured interviews with various system stakeholders such as members from the development team, Scrum Masters, software testers and business managers. The overwhelming response from the cohort of interviewees was that the Scrum of Scrums meetings did not work well and in some instances, these meeting were regarded as a wasted effort.

The Problems with Scrum of Scrums

The main problem with the Scrum of Scrum meetings was that the participants were not interested in what others were doing, thereby impeding the prospect of achieving synergy between participants at the meeting. Another aspect of concern was that the schedule of having weekly meetings which were deemed to be inadequate to deal with problem situations that were occurring on a daily basis and required immediate intervention. However, a significant outcome of this study is the report of success with regards to a scaled down version of the Scrum of Scrum meeting. The scaled down version is named the *Feature Scrum of Scrum* meeting that consisted of team representatives from teams that were jointly developing a specific feature/aspect of the system. In this situation, there was better synergy between the participants because they all had a better understanding of the specific feature that was being discussed. The empirical evidence presented in the Paasivaara *et al.* study converges to the conclusion that the strategy of using a Scrum of Scrums meetings to enhance the scalability of Scrum methodology does not work well in all circumstances. However, the ad hoc intervention with a scaled down version to remedy the shortcomings of the Scrum of Scrums proved to be more successful. The idea of scaling down the Scrum of Scrums approach to create a better collaborative environment between Scrum development teams is accorded greater coverage by Bradley (2014).

While Scrum has been highly endorsed for its substantive support for project management (Machado *et al.*, 2015), the methodology has been criticised for not having an equal focus on the engineering aspects of software development (Mushtaq & Qureshi, 2012; Ranasinghe & Perera, 2015). A significant absence from the methodology is any reference to test driven development (TDD) or code refactoring. According to Cao and Ramesh (2008), review meetings are not good enough for verification and validation and cannot replace the 'traceability' provided by specifying upfront acceptance tests. The upfront acceptance tests provide the customer as well as the development team with a source of reference to determine if the correct product is being built (validation) and whether it is being built correctly (verification). The suggestion of including TDD as part of Scrum methodology is also supported by Kniberg and Farhang (2008) as well as Li *et al.* (2010). In both these studies, it was found that the iterative nature of Scrum prioritised acceptance testing thereby improving customer satisfaction with the system and ultimately contributing significantly to the level of success of the methodology. However, the methodology is centered on the philosophy of ensuring a quick release of working software and 'reduced time to market'(Ahmed *et al.*, 2010). This places more stress and time pressure on the development team making them reluctant to engage in verification tests and code refactoring in order to improve the quality of system code and the maintainability of the system (Li *et al.*, 2010).

Scrum of Scrum Compromised by Lack of TDD

According to Kniberg and Farhang (2008), it is the lack of TDD that compromises the capacity of Scrum to produce quality code thereby diminishing the maintainability of the system. The assertion by Kniberg and Farhang was based on an experience report from the development of an online gaming software system that made use of Scrum methodology. Perfective[14] and adaptive[15] maintenance of the system was difficult to achieve because of the difficulty of understanding the code compounded by the issue of "…all the criss-crossing and circular dependencies riddling the code" (Kniberg & Farhang, 2008, p. 442). From a classical software engineering viewpoint, compromising the maintainability of a system is not economically viable. This assertion is based on the claim by Schach (2008, p. 13) that approximately 75% of the cost of software system development

[14] Perfective maintenance is defined by Schach (2008, p. 517) as a change made to code to improve its efficiency or enhance its functionality.

[15] Adaptive maintenance is defined by Schach (2008, p. 517) as changes made to a software system to react to changes in the operating environment of the system.

and maintenance is incurred in the post-delivery maintenance phase. Also, the widely accepted benchmark for software quality, named the ISO/IEC 9126 quality standard (see Jung *et al.* (2004)) has listed maintainability as one of the mandatory criteria that determines the quality of a software product. If Scrum methodology does not prioritise TDD, code refactoring and extensive documentation, then according to the ISO/IEC standard, the maintainability and quality of the information system being developed is bound to be compromised.

Agile and Code Quality

At the Agile 2006 Conference, Khramov (2006) informed delegates that code quality was not the main objective of agile methodology and there was no positive correlation between the quality of the code and the success of the system. According to Khramov, code quality is regarded as optional and the real goal of agile development is the commercial success of the product or the timely solution to an important problem where the focus is on time and cost benefit rather than quality of the system. Khramov's assertion regarding code quality and software success is based on an analysis of data from 80 software development projects and is commensurate with a similar sentiment by Wolff and Johann (2015) as well as Kanellopoulos and Yu (2015) that business return on investment with regards to software development is not exclusively dependent on the quality of the code.

The extent of the problem with regards to code quality is highlighted by Khramov's observation that poor code quality and erroneous software are not the main contributors to software failure. An elucidation of the concept of software success/failure is however, not a trivial one. As Paulk (2014) points out, the success of a software system is bound to its context of usage and may be driven by cost and schedule predictability as well as operational excellence (as is the case for software systems commissioned by government departments). From a commercial perspective, "…success is based on functionality delivered and the relationship of that functionality to business objectives" (Paulk, 2014, p. 3).

After conducting a comprehensive analysis of studies that examined factors that were deemed to be critical to the success of a software system, Söderland *et al.* (2012) posited that overall customer/end user satisfaction is the main criterion that determines success of a software system. It should be noted that the concept of customer satisfaction is vague, highly subjective and time dependent. Much of this vagueness has been operationalised, courtesy of the theoretical model of end user acceptance contained in the Technology Acceptance Model (TAM) proposed by Davis (1985) as well as the Information System Success Model (ISSM) proposed by DeLone and McLean (1992).

The End User Perspective

The end user perspective is a pivotal contributor to the success of a software system, and the iterative nature of agile development positions the methodology firmly in a vantage point to facilitate customer satisfaction and commercial success of the software system (Paulk, 2014). However, the allure of obtaining customer approval of the system thereby enabling a quick release of usable software detracts developers from the software engineering intrinsic activity of producing quality code. A situation that Kruchten *et al.* (2012) describe as placing the system into "technical debt" (p. 18) in the sense that software that is released early is bound to have flaws that will need to addressed at a later stage. Scrum methodology does not have an explicit focus on minimising the technical debt. The preceding deduction is aligned to a similar sentiment by Kniberg and Farhang (2008) and Liu *et al.* (2010) and may be seen as an impeding factor in ensuring the sustainability of a software system developed using Scrum methodology. The sustainability becomes tenuous when the system has to undergo perfective or adaptive maintenance because of the effort that will be incurred to change a system that has not been optimised for change. It should be noted however, that this is not a negative indictment on agile methodology in general. The engineering-like attributes of XP such as code refactoring, TDD, continuous integration and pair programming (in certain cases) enhances the prospect of enabling the production of better code quality in a software system (Khramov, 2006) thereby mitigating the risk of excessive technical debt.

Scrum has been established as the most popular agile methodology (Mundra *et al.*, 2013) largely based on its support for the project management aspect of software development (Mushtaq & Qureshi, 2012). The Scrum philosophy of ensuring a quick release of software so that the customer has the opportunity to interact with the system from an early stage thereby enabling an accurate capture of customer requirements has also been a pivotal contributor to the success of the methodology. Scrum has however been challenged for its limited support for ensuring quality system code thereby compromising the maintainability of the system. Also, the lack of comprehensive documentation and attention to better upfront system designs have cast a measure of doubt regarding the capacity of the methodology to handle the development of large complex systems or systems that serve a mission critical purpose.

2.4.6 Challenges Facing Agile Software Development Methodology

In terms of offering a better alternative to the plan driven prescriptive approach to software development, the increasing popularity of agile methodology has culminated in the establishment of the methodology as the de facto methodology for software development (Dingsøyr & Moe, 2014; Scheerer *et al.*, 2014). A possible explanation for this phenomenon could be traced back to Jackson's (1995) ontological analysis on the state of computing and his reference to the conceptual gaps of understanding between the technological realm and the societal realm and the need for software developers to bridge this gap. According to Boehm and Turner (2003), the agile philosophy of elevating the significance of individuals and interactions over processes and tools is a step in the right direction towards the quest to lessen the gap between the technology and the society in which the technology will be used. A pivotal strategy in this regard is the agile tactic of obtaining maximum input from the customer by suggesting the presence of an on-site customer to provide the development team with accurate user stories and to provide feedback on the evolving system at review meetings thereby ensuring a high level of customer involvement throughout the development life-cycle of the system. This is unlike prescriptive development methodologies such as the Waterfall methodology where customer requirements are established at the beginning of the development effort with very little recourse left to the customer to subsequently adjust the requirements specifications document in response to a volatile application domain (Abbas *et al.*, 2008).

The benefit of having extensive user involvement in the software development process is confirmed by Bano and Zowghi (2013) and Kundalram (2013) who reported on the positive correlation between user involvement and system success. Congruous to this finding, Morandini *et al.* (2017) refer to the imperative for software development practices that were observant of changing user requirements because of the dynamic nature of the social context in which these systems function. To a large extent, these requirements resonate quite well with many of the principles underlying the agile philosophy of software development. From a practitioner perspective, the allure of using a methodology that is adaptive and oriented towards satisfying user requirements has been instrumental in ensuring high adoption rates of agile methodology. The popularity of Scrum has largely been attributed to the resilience of the methodology to an unstable requirements elicitation phase. The adjustments that may made to the Product Backlog to factor-in new and changing user requirements is all part of the framework of development practices intrinsic to the Scrum methodology. The academic community has also accepted that agile methodology has generally been instrumental in improving the software process. There is however a concern regarding the lack of empirical evidence in the academic literature attesting to the success of the methodology and the lack of an integrative theory to underpin studies that analyse the success of the methodology (Abrahamsson *et al.*, 2009; Dybâ & Dingsoyr, 2008; Paulk, 2014)

The current discourse on agile software development has covered a spectrum of agile methods such as the strategy of enlisting an on-site customer to enhance development, pair programming, TDD and code refactoring, minimalist documentation and up-front system designs. These methods have been classified under 2 prominent agile methodologies named Scrum and XP. The distinctive strengths of Scrum is to enable better project management while XP provides software engineering guidance to enhance the quality of the coding effort. There is however, a unanimous acknowledgement that the agile methods are context-bound to the specific requirements of the project. A framework to guide the contextual applicability of agile methodology is provided in Table 2.4. This framework comprises of contributions by Boehm (2002), who provides an ideal scenario for the optimal implementation of agile methodology (named the 'Agile sweet-spot' in Table 2,4) and a counter scenario, (named the 'Agile bitter-spot' in Table 2.4) suggested by Kruchten (2004).

Table 2.4: Agile Sweet-spot (Boehm, 2002) and Agile Bitter-spot (Kruchten, 2004)

Aspect	Agile Sweet-spot	Agile Bitter-spot
System Specifications	Emergent requirements; rapid and late change to the requirements specifications is expected	
Type of project	New development projects	Maintenance projects
Project Duration	Shorter development timeframe; 2 to 3 months	Long term project spanning up to 2 years
Location of Development team	Developers need to be knowledgeable about the process, co-located and collaborative	Development team works in a distributed environment
Size of development team	Development team is small; 15 to 20 developers is optimal	Large development team; excess of 200 developers
Customer	There is a core need to have a dedicated, on-site customer who is representative of the application domain	The lack of a representative, on-site customer
Refactoring and Documentation	Refactoring and documentation should not incur major overhead	A system that needs extensive documentation to faciltate continuity and communication between team members

Architecture and Primary objective	A minimalist approach to upfront system architecture; objective is to meet an immediate need; not much focus on low level architectural issues.	System is designed for stability and long term maintenance; comprehensive upfront design models are required; expansive upfront detail is expected

According to Turk *et al.* (2014), knowledge of the context of application for agile methods is pivotal in order to maximise the value obtained from agile methodology. Aligned to this claim, Turk *et al.* conducted an analysis of the assumptions underlying agile methodology in order to generate a list of conditions that provide guidance with regards to the applicability of agile methods. The conditions identified in the analysis conducted by Turk *et al.* are congruent with the listing in Table 2.4. The notable addition to this list is a reference to the limited support that agile methodology provides for the development of safety-critical software. This claim is based on the minimal focus on formal software engineering techniques (such as formal specifications, rigorous code path testing, extensive documentation, quality assurance and continuous redesign) in the underlying assumptions of agile methodology.

What has emerged from the preceding discussion is that the successful implementation of agile methodology is intrinsically linked to its context of implementation. In this regard, practitioners have been reliant on anecdotal evidence that is based on intuition and experience reports to develop conditions to provide this guidance (e.g. Boehm, 2002; Kruchten, 2004; Turk *et al.*, 2014). This set of heuristics serve the purpose of providing an informed underpinning to the implementation of agile methodology in order to enable practitioners to obtain immediate benefit. Paulk (2014) does however, warn against the temptation of using these agile heuristics to create a piece-meal variant of XP or Scrum in order to suit the application domain. Such an adaptation should be done on the basis of empirical studies that provide a reliable guide for an informed implementation of the methodology. However, in order to extend the applicability of the methodology to domains where it has been perceived as being inappropriate, a formal unifying framework that incorporates the assumptions underlying the methodology as well as the contexts in which it is deemed appropriate for implementation, is required.

2.4.7 The Quest for a 'Theoretical Lens'

In an article for the Software Development Times, West (2015) makes an erudite acknowledgement of the importance of understanding the context within which a software development methodology is used. The opinion expressed in the article alludes to the imperative for software development teams to implement a software development methodology that is befitting of the circumstance in which it is used. This enables practitioners to leverage the advantage of using preferred tools and expertise in order to maximise the chances of producing a successful system. Whilst the afore-mentioned strategy is condoned, there has to be a unifying framework that incorporates these "success situations" into a repository of knowledge that may be used as a resource to guide future software development projects. According to West (2015), the consequence of not having a unifying framework that integrates context and methodology is that there will be a vast amount of "siloed information" (p. 1) on the software development process that will be fragmented and severely lacking in cohesive support for the development process. The practitioner perspective on this matter is congruent with the academic opinion that there is a need to understand agile methods within its context of use in order to optimise the benefits that may be gained from using the methodology (Abrahamsson et al., 2009; Dingsøyr et al., 2012; Kirk & MacDonell, 2014). However, this understanding needs to be underpinned by an integrated framework/ theory that informs the use of the methodology in different project and organisational contexts. The call to make use of better theoretical frameworks to extend the relevance of scholarly contributions on the topic is eloquently encapsulated in the statement by Dingsøyr et al. (2012) that the pioneering contributions on agile methodology have "…established a foundation on which the edifice of software development theory and practice can be built" (p. 1219).

Aligned to the call by Dingsøyr *et al.* for a "robust theoretical scaffold" (p. 1219) to underpin further research on agile methodology, Kirk and MacDonell (2014) suggest that the development of theory on the use of agile methodology should be centered on the relationship between the implementation of the methodology and context in which it is used. An outcome of this understanding is that the academic community will be in a better position to provide a framework that informs the implementation of the methodology in a professional setting. (Kirk & MacDonell, 2014). This imperative to underpin the implementation of agility with a theoretical basis should however, not be seen as an attempt to streamline the process thereby resulting in a paradoxical situation where the core principles of agility are eroded by a framework that is perceived to be prescriptive. The theoretical intervention needs to embrace the multi-faceted and contextual nature of software development (Lyytinen & Rose, 2006) so that organisations have at their disposal an academic frame of reference that may be used as a platform/cohesive body of knowledge to guide the adoption and adaptation of agile methodology (Abrahamsson *et al.*, 2009).

Such an intervention will enhance the possibility of extending the applicability of agile methods, which are traditionally associated with small, non-critical systems where development teams are co-located and user requirements are elicited dynamically. According to Abrahamsson *et al.* (2009), many studies have reported on the issues pertaining to the adoption of agile methodology (AM). However, this knowledge needs to be encapsulated into a theoretical framework that will enhance the prospect of meaningfully engaging in post-adoption studies that examine the sustainability of AM. An example of such a study is the one conducted by Port and Bui (2009) who studied the viability of using a mixed methods approach that entailed an integration of AM and a plan based (PB) methodology to develop software. The study used a simulation strategy to vary the complexity of the software system. An outcome of this study is that the approach of mixing AM with PB methodology is confirmed as a viable option to mitigate the risks (such as the lack of architectural stability (Cao *et al.*, 2009; Yang *et al.*, 2016) imposed by using AM to develop large, complex software systems. The Port and Bui (2009) study represents an initial incursion into the realm of extending the applicability of AM. The initiative to extend the applicability of AM was sustained by Cao *et al.* (2009) who conducted a multi-site case study to determine how AM may be adapted for use in different contexts. The study uses adaptive structuration theory (AST) to provide an adaptation framework for AM based on the requirements of different projects and organisational environments. AST is a framework proposed by DeSanctis and Poole (1994) that attaches greater priority to the social aspects of technological interventions rather than the technical aspects of the intervention. The Cao *et al.* study used AST to understand the adaptation of agile methodology as a consequence of the social interaction that occurs when the methodology is used. A significant aspect of the study is the presentation of empirical evidence attesting to the need to apply specific tenets of agile methods for the varying contexts of usage. A corollary of this finding is that it is not viable to apply agile methods in their entirety and there is a need to temper agile methods so that there is a strong alignment with the prevailing organisational culture with specific focus on priorities established by higher level management and the development styles of software teams and the type of project that is being undertaken. The study by Cao *et al.* (2009) provides the empirical support for the claim by Nerur *et al.* (2005) that agile development is characterised by social interaction where the various stakeholders, including business analysts, developers, project managers and end users engage who engage repeatedly in a reflective mode and leverage their experiences of using the methodology to curate a customised version of the methodology.

The reference to the various stakeholders involved in software development in an organisational context opens up another dimension to agile software development (ASD) that has largely been neglected in the literature review this far. This is a reference to the role that ASD plays from an organisational/enterprise-wide perspective. While the academics and practitioners have devoted a lot of attention to the operational issues regarding ASDM, the wider environmental impact has not received much focus in the literature on ASD prompting Fitzgerald and Stol (2015) to suggest that any attempt to adapt ASD will be futile if it is not done from an enterprise-wide perspective that incorporates business objectives. Based on the preceding argument, an incursion into the enterprise-wide impact that ASD will incur is warranted.

The business dimension usually manifests in respect of the cost to develop software as well as the time and resources consumed and invariably the quality of the software produced (Basili *et al.*, 2013). In order to address the issue of business interests, a traditional practice by project managers was to make use of a command and control strategy to uphold the business imperative (McAvoy & Butler, 2009). However, the rigidity inherent in such a dictatorial approach is not commensurate with the principles of agility. From an agile perspective, project managers are expected to provide an environment that facilitates participatory decision-making thereby devolving authority to all members of the development team. In order to recognise the impact of the afore-mentioned social intervention, there is a strong imperative to make use of a socially oriented theoretical base such as that provided by AST in studies that purport to obtain a deeper understanding of agile methodology.

2.5 The Enterprise-Wide Context

According to Ambler and Lines (2012), agile software development teams do not work in a vacuum. There has to be a sense of "enterprise awareness" (p. 17) that is integrated into the software development process. Ambler and Lines are of the opinion that agile software development methodology has many proven benefits exclusively from a software application perspective. However, it does not handle the complexities inherent in the activity of integrating the software applications into the organisational IT infrastructure. These sentiments are quite controversial in the sense that the underlying philosophy of ASDM is the concept of simplicity and the adoption of lightweight protocols in the development of software. In a rebuke of these attempts to undermine the philosophy of agility, the contributory authors of the Agile Manifesto, used the GOTO 2014 Conference platform (see Fowler *et al.*, 2014) to explain that any attempt to obfuscate the simplicity in ASDM will impede the progress that the software development community has made with regards to the development of software systems that are delivered on time and meet user requirements. West (2015) concedes that the introduction of agility into the software process has achieved much success.

However, the adoption of agility is constrained by organisational culture and governance over the software process.

Agile and Business Value

The imperative imposed by most organisations is that software systems need to uphold the priority of delivering business value. In order to achieve this, software development teams need to facilitate systems development so that there is complete compatibility with the existing IT infrastructure. West is also critical of the highly touted agile based strategy of frequent release of systems. The idea behind this strategy is that feedback can be obtained early, enabling the development teams to handle arising issues. However, this is not easy to achieve because maintaining an IT infrastructure that supports dynamic, flexible releases may not be practically feasible because of the business oriented controls that may impede such an initiative. Hence, there is a break in the lineage between business value, software development and the release of the software systems into a production environment. In order to address this impasse, many experts in the software engineering community have rallied around the concept of a complete life-cycle model for software development. The idea advocated is that the activity of software development has to be contextualised from an enterprise-wide perspective, rather than just a software development perspective. In order to achieve this enterprise-wide focus, new software process models have been proposed so that agile methodology may be scaled to be compliant from an organisational/ enterprise-wide perspective. These process models are discussed in the subsequent sections.

2.5.1 Water-Scrum-Fall

The Water-Scrum-Fall model resurrects the Waterfall approach, from a holistic perspective (West, 2015). The main elements of the Waterfall approach is the upfront establishing of requirements, the analysis, design and construction phase and finally testing and maintenance. The newly proposed model retains the sequential structure of the Waterfall model, in the sense that the first activity is to establish the business value, requirements and plans for the system, followed by a development phase (where Scrum has been endorsed as the driving methodology) and finally the phase where the system is released into a production environment where it is used by the customer/ end user. This sequence of activities is named the *Water-Scrum-Fall* methodology for software development.

The resurrection of the diluted version of the Waterfall approach was subjected to scrutiny by many within the software engineering research community (e.g. Aitken & Ilango, 2013; Bannink, 2014; Theocharis *et al.*, 2015; West, 2011) The main outcome in all of these studies is that the transition to Agile Methodology and Scrum in particular has not been a smooth one. The main reason for this phenomenon is that the entire organisation is not willing to make a transition to an agile operational mode, resulting in a Water-Scrum-Fall methodology that serves the dual objectives of maintaining traditional organisational processes and also embracing an innovative development culture.

There is however a perceived lack of empirical evidence to truly understand the adoption of Agile Methodology in the context that it serves (Bannink, 2014; Theocharis *et al.*, 2015). Much of the literature focuses on the internal workings of the methodology, whilst ignoring the organisational processes that form the environment in which agile methods operate (Dingsøyr *et al.*, 2012). These sentiments are re-affirmed in the systematic literature review conducted by Theocharis *et al.* (2015) that examined 473 papers that investigated the use of agile methods in an organisational setting. The significant outcome from this study is that most organisations have to improvise and develop ad hoc solutions to compensate for the lack of support that Agile Methodology provides for the 'organisational interface'.

Agile methods have a strong system development focus while the traditional approaches do incorporate a specific phase to faciltate compatibility between the newly developed systems and the interfaces that enable integration with the enterprise. The 'organisational interface' shortcoming of the Agile Methodology has been a catalyst for the popularity of a hybrid approach. A hybrid arrangement between agility and a plan-driven methodology such as the waterfall approach is beneficial from a dual perspective. The plan-driven approach provides a controlled environment with development phases that are dedicated to transition of the newly developed system onto the organisational infrastructure and the agile approach enables a dynamic, shorter development cycle that enhances the evolving system's visibility so that end user feedback is obtained a lot quicker.

Theocharis *et al.* (2015) adopted a case study approach where the use of agile methods is investigated in an organisation that recently made a transition from traditional software development methodology to a Scrum based methodology. The results of the study show a trend where the development teams had a preference for a hybrid methodology that comprised of the routine Waterfall processes like intensive upfront analysis and design followed by Scrum based development. The final phase entailed a software release process that was described as infrequent and time consuming due to the effort required to integrate these system releases with the organisational IT infrastructure. This methodology falls under the classification of 'Wagile' development, an umbrella term used to describe a methodological approach that is planned as an agile approach, but has a tendency to revert back to the implementation of Waterfall methods.

The apparent gravitation of organisations towards a Water-Scrum-Fall/Wagile approach under the 'alleged' claim of full agility is as a consequence of the inability of Agile Methodology to achieve its objectives at enterprise level. The potential for the Water-Scrum-Fall/Wagile methodologies to handle organisational interfacing issues were still not perceived as adequate, a shortcoming that prompted the conception of 3 enterprise-oriented versions of agile methodology named DevOps, Disciplined Agile Delivery (DAD) and Scaled Agile Framework (SAFe), which are all discussed in the subsequent sections.

2.5.2 DevOps

A 'spin-off' or an extension to agile software development is the strategy referred to as DevOps. Aligned to the agile strategy of ensuring quick software release, and implementation, DevOps is a strategy that attempts to reduce the 'disconnect' between the developers (Dev) and the operators (Ops) of a system (Limoncelli & Hughes, 2011). It should be noted that although the DevOps concept has been conceived around 2009 (Kim, 2013), it is currently at a stage of infancy with regards to the rate of adoption (Zhu *et al.*, 2016). DevOps is a concept that attempts to dismantle the 'silo-based' or fragmented approach to application development and the delivery and operation of the application from an enterprise perspective (Ravichandran *et al.*, 2016). Traditionally, the development of an application proceeds linearly from development/coding to quality assurance to integration of the application with an organisation's IT infrastructure. This path entails the involvement of software developers, quality assurance (QA) personnel and IT management who are entrusted with the task of providing a smooth operational environment (Ops[16]) for the successful implementation of the application. The DevOps philosophy comprises of a strategy where the specialists who are involved in each of these individual activities are brought together to work in a collaborative environment (Ravichandran *et al.*, 2016). It mitigates a situation where developers write the code and entrust the responsibility of deployment of the application onto the operations staff. DevOps is an initiative to embrace an approach to software development and deployment that integrates the different silos of the IT department that typically are involved in the software development, deployment and maintenance of an application in an organisation. A closer working relationship between developers and operations staff will enable the ongoing management of the application to be conducted in a manner that enhances the prospect of a quick and efficient deployment of the application to a 'live environment' as well as a quick turnaround time when it comes to issues of perfective and adaptive maintenance. Hence, DevOps embodies a working environment that prioritises collaboration, cross-functional teams and enables early and continuous delivery of working software. The preceding statement reflects the resonance of the DevOps concept to the principles of ASDM.

If one had to adopt a restricted view of the DevOps concept, then there could be a claim made that the linkage between DevOps and ASDM is a tenuous one. This assertion is based on the perception that much of the deliberations regarding

[16] The operators (Ops) of a system include organisational personnel who have any form of contact with the system after it has been released by the development team as a finished product or a finished version of a product. The list includes network administrators, database administrators, system administrators, network engineers, security engineers and general application support staff.

ASDM has been conducted from a coding/purely software development perspective while the operational environment in which the application delivers its expected functionality is not the domain of the development team. However, Mueller (2016) does make the claim that if the development team did not take cognisance of the operational side of the system, which becomes the main focus during the deployment phase, then the benefits of having adopted an agile approach will not be realised. The speed and agility used to facilitate competitive advantage via the 'quick-release' of software may become counter-productive if the development team did not consider issues pertaining to systems integration as well as compatibility with the technological infrastructure. According to Mueller, cognisance of these issues need to take place at high level systems development planning meetings as well as during the deployment phase once an initial version of the system has been released. This close collaboration between the development team and the operational staff embodies the DevOps framework. Sharma (2017) provides an overview of this close collaboration intrinsic to the DevOps approach by suggesting that:

- developers have to work with operations staff so that they can understand the environment in which the systems work;
- operations staff need to be close observers of the development process so that they have an intimate understanding of the requirements as well as the coding logic used.

Whilst these characteristics of the DevOps approach are suggestive that the DevOps strategy is easily understood, many authors (e.g. Bass *et al.*, 2015; Roche, 2013; Sharma, 2017) allude to the difficulty of providing a precise definition of the DevOps strategy. In many cases authors are willing to propose a rather informal description of the DevOps strategy such as Limoncelli and Hughes (2011) who provide an uncomplicated interpretation by advocating that DevOps represents a strategy that brings developers and operators closer together. Mueller (2016) adds to this interpretation by suggesting that the developers and operations staff collaborate on a project throughout the development and service lifecycle. This collaboration comprises of an integration of Dev and Ops functions from design through to the development process up until production and support for the system. This collaborative strategy blurs the traditional distinction between development, quality assurance and operations. It also has implication from an organisational culture perspective because it requires the various stakeholders to work in an interactive manner to facilitate the building, testing and release of software in a quick and reliable manner.

The DevOps strategy as outlined in Mueller (2016) requires that once a development team declares that a specific version of a system is ready to be deployed, the assumption is made that any further development will be suspended while the application is deployed into production. At this juncture, the application is subjected to 'live' testing and intensive scrutiny whilst in the 'live' environment. The DevOps practice requires that developers are allocated the task of observing the progress and analysing systems errors so that remedial action can be taken. In this way an iterative relationship is maintained between the developers of the system and the operators of the system. This iterative arrangement enables quicker releases and the implementation of quicker changes that may be required by the operators who enjoy the benefit of having immediate access to the developers.

2.5.3 Disciplined Agile Delivery (DAD)

According to Ambler and Lines (2016), many organisations adopt agile methodologies such as Scrum because it is the best strategy to provide guidance to software teams with regards to the coding aspect of the application. However, the 'beauty' inherent in these agile methods is lost because the methodology does not provide adequate support for the full life-cycle of the application from an organisational perspective. The criticism of agile methodology in its current form stemmed from the perceived inability of the methodology to handle the release of the solution that has been developed into a production environment. Ambler and Lines do concede however, that Scrum methodology had achieved substantial popularity and success from a purely development perspective. It was the operational side that needed attention. In order to address this situation, Ambler and Lines proposed a modified version of Scrum in 2012. The modified version of Scrum is referred to as the Disciplined Agile Delivery (DAD) approach (see Ambler and Lines (2012)). DAD is an extension of agile methodology (Scrum in particular) where the focus is on ensuring that the solution provided by agile teams is successful at an enterprise level. In order to achieve this, Ambler and Lines leveraged the best practices from Scrum, XP and the Unified Process to propose a methodology that shifts the focus to application delivery, operation and support from an enterprise/organisational context. The preceding narrative is echoed in the comment by Ambler and Lines (2012, p. 9) that:

> *Core agile methods such as Scrum and XP are typically project focused, whereas DAD explicitly strives to both leverage and enhance the organizational ecosystem in which a team operates.*

Essentially, the DAD methodology re-aligns the focus from producing software to providing an IT solution that resonates with business, technical and the cultural constraints in which that solution operates. An overview of the DAD methodology is provided in Figure 2.13.

Inception	**Construction**	**Transition**
(*One or more short iterations that entail requirements, modelling, release planning and acquiring stakeholder consensus*)	(*Identify highest priority work items; Many short iterations (Scrum based) to service iteration backlog in order to produce a potentially consumable solution after each iteration; demonstrate solution to stakeholders; obtain feedback)*	*(Release solution into production; operate and support solution whilst in production mode enabling evolution into final product)*

Figure 2.13: The Disciplined Agile Delivery (DAD) lifecycle model (Ambler & Lines, 2012, p. 12)

The underlying philosophy of the DAD methodology is to provide sufficient guidance, but not to be overly prescriptive. The Inception phase as illustrated in Figure 2.13 may be seen as an ‘envisioning’ phase where the system’s evolution is mapped out to the developers as well as the stakeholders. A significant activity in the Inception phase is to set up a development environment that facilitates quick and easy release of the application into production. Also, an initial plan of the application release schedule is drawn up together with an architectural/design model that provides a logical view of the application so that there is an alignment between the objectives of the application and the business/organisational objectives. The activities that have been listed are regarded as goals of the Inception phase and there no prescribed way of achieving these goals. The rationale for this approach is that the development teams are at liberty to customise the development processes in order to address the context of the situation in which the application is being developed

The goals of the Construction phase are to produce a 'demonstrably consumable solution' that addresses stakeholder's requirements and has an 'organisational fit'. This is achieved by employing techniques such as continuous integration, developer regression testing and test-first development. The actual development is executed by implementing all of the ceremonies intrinsic to the Scrum methodology. The main point of departure from traditional Scrum is that the focus is on ensuring that the solution is compatible with the existing architectural framework that underpins the organisation's IT infrastructure.

According to Ambler and Lines (2013), the lack of enterprise-wide focus is one of the reasons that popular agile methodologies such as Scrum were not fully successful. In an article titled "Going Beyond Scrum", Ambler and Lines make the point that agile teams do not work in isolation and application solutions produced by Scrum teams should be regression tested so that it is compatible with existing organisational processes, is compatible with the data infrastructure and compliant with security and usability constraints that have been established as an organisational norm (referred to as the *organisational ecosystem* by Ambler and Lines). In order to develop solutions that have an 'enterprise-wide' awareness, Ambler and Lines make several suggestions that collectively form the essence of the DAD framework that they propose as an extension/supplement to Scrum methodology. The underlying strategy of DAD is to arguably ensure that the Scrum team works closely with enterprise professionals. The reference to enterprise professionals is where the organisational linkage is established.

According to Ambler and Lines (2013), ‘enterprise professionals’ is a reference to personnel in the organisation who ensure that business processing protocols are maintained and upheld by new and emerging IT systems. These include IT based personnel who oversee aspects such as IT governance database design and administration, IT security and user interface design and quality control and testing. The close collaboration with enterprise professionals and operations staff is representative of a DevOps philosophy that has been “…baked right into DAD” (Ambler & Lines, 2013, p. 11).

The goal of the Transition Phase of DAD is to ensure that the system’s stakeholders have worked with the new application and are *delighted* by its’ performance and conformance within the *organisational ecosystem.* Ambler and Lines make a claim that the DAD framework ensures that the Transition Phase is a smooth one. This is in contrast to the current, traditional agile situation where transition and deployment of newly developed systems is where the major bottleneck to agile application delivery is experienced (Ambler & Lines, 2016). The smooth passage for the Transition Phase is facilitated and enhanced at the Construction Phase where there is greater stakeholder involvement from a training and consultation perspective. This strategy has a strong resonance with the DevOps approach.

2.5.4 DAD Acceptance

From a rational and pragmatic perspective, the DAD framework makes a lot of sense. As suggested in Ambler and Lines (2012), one of the intentions of DAD is to provide an agile based methodology for software development that has an enterprise-wide focus. The DAD approach enhances the scalability of agile methodology so that the methodology has the ability to handle the development of large scale applications in an organisation.

The advent of DAD may be perceived as a relatively recent contribution to the domain of agile software development, thereby compromising the possibility of current widespread acceptance and usage of the methodology. However, the DevOps mentality that prevails in DAD paves the way for the methodology to gain traction in the software development domain. This assertion is supported by the empirical evidence provided in the 11th Annual State of Agile report compiled by the software development company named VersionOne. The survey was conducted in 2016 (see VersionOne, 2016) and is regarded as the biggest global survey of agile usage behaviour. A significant outcome of this survey is that Scrum is the most widely used agile development methodology and almost 71% of the respondents have engaged or intend engaging in a DevOps initiative where the focus is on enterprise-wide solution development and delivery. It should be noted however, that DAD has a very low report of usage (less than 1%). This phenomenon may be explained by the understanding that DAD is a framework that is super-imposed onto Scrum methodology and is not seen as a methodology by itself. Hence, it may be difficult to obtain a concise perspective on the usage of DAD. However, the conceptual acceptance of DAD superimposed onto Scrum is being manifested in the DevOps strategy.

2.5.5 The Challenges to the DevOps Strategy

The theoretical foundation of DevOps as discussed in the preceding section has an aura of acceptability and viability. In essence it sounds very good in theory. However, as cautioned by Kerzazi and Adams (2016), the DevOps concept is currently at a stage of infancy and lacks a common vocabulary and a substantial body of knowledge as well as empirical evidence attesting to its success. The roles that may emanate from a DevOps strategy is rather vague and organisations do not have an understanding of the skill-set required by a DevOps engineer. From a technical perspective, the DevOps approach is strongly aligned to Scrum methodology but there is lack of engineering-oriented detail that specifies how DevOps provides an enabling environment for operations processes within the confines of a Scrum methodological framework (Vaidya, 2014). The Scrum methodological framework currently does not have any reference to operations activities or the roles played by operations staff during the development process. From a social perspective, Sharma (2017) cautions that DevOps requires an organisational-wide mind-set change that may be seen as a business processing re-engineering initiative and its success depends on support from upper level management in an organisation. The DevOps philosophy attaches the highest priority to ensuring that there is a collaborative environment that enables seamless transition of processing requirements from business to development to operations. The DevOps strategy is based on an untested assumption that such a collaborative environment may be easily achieved. As Riungu-Kalliosaari *et al.* (2016) caution, the influence of organisational culture should not be underestimated and it may have an impeding effect that will prevent such a collaborative environment from occurring naturally.

One of the biggest issues with the strategy of DevOps is that it enables a culture of continuous deployment (CD) of working software. While this may be seen as a positive attribute and has a strong resonance with agile philosophy, it could also become severely delimiting in the sense that it creates an IT environment that is always in a state of transition. The release of new features for a system needs to be carefully planned and managed so that the users of the system are not overwhelmed with too much change in a short time period. The irony of the 'pre-DevOps' phase of agile development is that the backlog and time delay caused by integrating new features into an existing system from an operational perspective has some unintended benefits. The time delay between development and deployment provided users of the 'old' system with a bit of a 'breathing space' to establish familiarity with that system before being faced with the task of getting to know the newly added features to the system. However, there are instances where a quick release of new features is pivotal to enable organisations to obtain competitive advantage. This is the case with Internet based organisations such as Facebook, Netflix and Etsy where DevOps has been used to increase the prospect of presenting customers with new features on a regular basis that enhance the quality of the interaction with the company's website (Shahin *et al.*, 2017).

In order to leverage the benefits of DevOps and facilitate continuous delivery and deployment of working software, it is essential that the Dev and Ops teams do not form silos and are easily accessible to each other. This strategy has been successfully implemented at Facebook (Savor *et al.*, 2016) in a rather extreme version of the DevOps concept. The strategy used at Facebook is to focus on the development of relatively small increments of functionality and enable the rapid deployment of the new features. In some instances, new features are added onto the main system in a matter of two hours. In order to achieve this rapid deployment, cross-functional software development teams are formed and they take full responsibility for the design, development, testing and configuration of the updates as well as support for the updates after they have been deployed. In the event of a problem, there is a single point of contact and that is the development team that was responsible for the new feature that has been added to the system. One of the main issues of the strategy of quick deployment is that it may work for organisations where the consequence of code failure or bugs in the code is not 'mission critical' and may be reversible. While it may be conceded that recovery will be quick because of the close collaboration between the deployment and development personnel, this benefit is appreciated in a context where the consequence of a flawed or incorrect system transaction may be reversed.

Based on the case study of Facebook and OANDA (an online trading organisation) Savor *et al.* (2016) report that in order for a DevOps strategy to work so that continuous deployment can take place, there needs to be a business process re-engineering effort that permeates throughout the organisation. This entails an organisational cultural shift that primarily requires full commitment by senior management and the main focus of the shift is that there has to be a drastic reduction in the layers of bureaucracy that impede the deployment of software updates to the main organisational system. In the case of OANDA, the entire management team that consisted mainly of business minded people were replaced by a management team that consists of people who have a software engineering background. This was done to facilitate a change in the management style from a hierarchical management structure to a more 'flattened' structure where innovation was prioritised at the expense of business accountability.

2.5.6 The Scaled Agile Framework (SAFe)

DAD and DevOps are representative of methodologies that address the weakness of agile methodology to scale to an organisational level. These initiatives are further extended by SAFe where the objective is to provide guidance on the implementation of agile methodology at enterprise/organisational level. As Dybâ and Dingsoyr (2008) point out, the implementation of agile methodology in larger organisations is challenging from a co-ordination and cultural perspective. Co-ordination becomes an issue when there is a greater number of stakeholders involved and multiple teams work on a single project. There is also the dimension of organisational culture where there is a natural resistance to change from a behavioural perspective. In larger organisations, this situation tends to get exacerbated and successful agile adoption requires a change in the entire organisational culture (Chandra Misra *et al.*, 2010).

In an attempt to address the issue of agile scalability from an enterprise-wide perspective Leffingwell (2007) introduced the methodology of SAFe that is underpinned by 4 different frameworks each configured to handle specific organisation environments. An overview of the SAFe frameworks (see Scaled_Agile (2017)) is provided for reference.

- Essential SAFe – Consists of a new structure named the Agile Release Train (ART) that functions at the lower software development level (called the SAFe Team level) and at a higher business and infrastructure level (called the SAFe Program level). At the team level, SAFe provides guidance on the coding part of system development. At the program level, SAFe provides guidance on the operations activities that enable business value. The core "ingredient" to the Essential SAFe is the ART that comprises of a cross functional team that delivers the development and operations value streams;

- Portfolio SAFe – An enterprise-wide plan that makes use of value streams (a term used to describe an enterprise-wide strategy to develop products, services or software systems) that provide value to the customer. The Portfolio SAFe is aligned to the organisational imperative to identify strategies that enable product differentiation in the marketplace and to ensure competitive advantage. Leffingwell (2010, p. 43) refers to these strategies as "a set of investment themes". These investment themes are achieved in the form of "epics" which is a term used as a high level descriptor of customer need and translates to a software development initiative. These epics are maintained in a portfolio backlog. One of the key role players is the Enterprise Architect, a person (or group of persons) who manages the portfolio backlog and works across programs (from the Essential SAFe) to provide technical direction that can arguably ensure that the outcomes for the portfolio are optimally achieved. The portfolio SAFe is a scaled up, business version of Agile Software Development Methodology;

- Large Solution SAFe – used for developing complex enterprise wide solutions; typically used for government and military systems and require multiple ARTs;
- Full SAFe–a SAFe configuration that is the most comprehensive version of the framework and provides support for organisations that build and maintain large, integrated solutions and require extensive collaboration across the organisation to include stakeholders that function at the SAFe Team, Program, Large Solution and Portfolio levels of the framework.

SAFe seen as the "Big Picture" Approach

SAFe provides a framework to guide the software process from a team and organisational perspective thereby reducing the divide between the business imperative and software development at the agile team level (Vaidya, 2014). Leffingwell (2010) calls this the "big picture" (pp. 32-33) approach to software development that has the objectives of providing an enabling environment for the achievement of business value as well as ensuring that there is sufficient collaboration between the various "pods of agile teams" (p. 35) that traditionally function in a disparate manner. This holistic approach to software development where agile development is contextualised from an organisational perspective and not just a software development team perspective, is highly endorsed by Fitzgerald and Stol (2015) as well as (Vaidya, 2014). Fitzgerald and Stol (2017) suggest that a framework such as SAFe provides the linkage between business, development and operations. The collaborative environment espoused by SAFe also reduces the "architectural decay" or "technical debt" (p. 9) incurred by many agile teams when there is no effort made to faciltate compliance of the evolving system with organsational architectural/infrastructure requirements. The imperative to ensure that deliberations regarding the scalability of software development methodology is given high priority is also highlighted by Boehm (2011) who provides a scalable version of the Spiral methodology for software development that is named the Incremental Commitment Spiral Model (ICSM). The main difference between the ICSM and the original Spiral model is the inclusion of and Operations and Production phase. The ICMS has a similar orientation to the Essential SAFe.

The Alignment of SAFe to Agility Principles

Theoretically, the SAFe framework embraces agile principles to provide an all-encompassing solution to the problem of the lack of scalability of agile methodology to an organisational level. Dikert *et al.* (2016) do however caution about the lack of academic research to verify the long term viability of comprehensive frameworks for software development such as SAFe. The main concern expressed is that the adoption of organisational-wide frameworks require a major change in the organisational norms when it comes to software development.

Agile, SAFe and Organisational Culture

One of the challenges faced in the transition to *basic* agile development was the issue of organisational culture. The adoption of agile methodology requires a shift in the organisational culture that is not easily achieved. A further imposition of agile methodology at the organisational and operational level (as espoused by SAFe) makes this transition a lot more difficult to achieve (Fitzgerald & Stol, 2017) resulting in only a lightweight adoption of SAFe at the Essential SAFe level (Vaidya, 2014). As Dikert *et al.* (2016) point out, a formal intervention to achieve agile scalability will require comprehensive staff training and support from senior management. The greatest obstacle to enable a framework such as SAFe is the 'top down' management style that will have to prevail to ensure that there is sufficient cooperation at all levels of the organisation to enhance the adoption of such a framework. During the transition from an 'old way of working' to the new framework, any problem encountered has the potential to be magnified because of people's general resistance to change and preferring to revert "…to the ways they know" (Dikert *et al.*, 2016, p. 97).

Organisation-Wide Agility

A further issue that compromises the attainment of organisation-wide agility is that of communication and coordination. In a multi-case study by Eklund *et al.* (2014) that spanned the banking, telecommunications and insurance sectors, it was found that scaling agile teams to an organisational level was not easily achieved. One of the main reasons for this phenomenon was the lack of coordination between Scrum teams that were co-dependent[17] resulting in a disjointed development effort. In order to alleviate this situation there was a need to appoint an oversight manager who is able to coordinate the activities between the various teams. Conceptually this adds another layer of management control thereby exacerbating the complexity of the development process and also compromising the agile principle of 'simplicity', prompting Thomas (2015), one of the co-authors of the Agile Manifesto to suggest that SAFe is not agile.

The discourse on software development methodology and the scalability of the methodology to an organisational level converges to a viewpoint that the organisational culture and social factors are pivotal enablers in the adoption of a software development methodology. The intertwining of software development methodology with the social realm necessitates an incursion into the essence of organisational culture. This 'digression' is perceived as crucial so that any empirical study to understand the adoption of software development methodology is cognisant of the influence of organsational culture (Sheffield & Lemétayer, 2013).

[17] A reference to agile teams that have a dependency on other agile teams for lower level functionality

2.6 A Discourse on Organisational Culture (OC)

The incursion into OC culture is necessitated in order to ascertain whether the concept of OC can be quantified so that the abstractionism inherent in this concept could be given a tangible form thereby enabling better comprehension of the organisational context. However, according to Leidner and Kayworth (2006), providing a precise definition as well as a strategy for measuring an amorphous phenomenon such as OC is one of the biggest challenges facing IS research. This drawback may be attributed to the multi-dimensional nature of OC (Simberova, 2015) or the lack of consensus regarding a precise definition of OC (Hu *et al.*, 2012). In such instances, a viable approach would be to follow the research design that is informed by the methods and suggestions of pioneers and respected writers in the domain of OC theory. From an OC perspective, van Muijen and Jaap (1999) suggest using the methods of Geert Hofstede and Edgar Schein as a point of reference.

Organisational Culture as an Abstraction that needs Acknowledgement

The seminal contributions made by Edgar Schein with respect to OC positions him to advocate a possible definition of OC. According to Schein (1985), OC is defined to be:

> *A pattern of shared basic assumptions that a group has learned as it solved its problems of external adaptation and internal integration, that has worked well enough to be considered valid and therefore, to be taught to new members as the correct way to perceive, think, and feel in relation to those problems* (p. 4)

Schein does warn however, that OC is an abstraction that needs to be respected because the influences that are created from the interplay between social and organisational relationships derived from culture can be quite overwhelming (Schein, 1983). From a software process improvement perspective, Schein's theory on OC is suggestive of a natural organisational tendency to preserve a traditional approach at the expense of embracing an innovation that changes behaviour and could possibly yield better quality. The resilience to change is extended to a point where traditional behaviour begins to be taken for granted and simply becomes unconscious assumptions that are taught to newer members of an organisation as a reality that should not be challenged because it is perceived as the proper way of doing things. These sentiments resonate quite well with the contributions by Gershon *et al.* (2004) and Simberova (2015) who suggest that OC is an embodiment of the norms, values, and basic assumptions of a given organisation. Much of Hofstede's contributions with regards to OC have a similar inclination, although he also conveys the disclaimer that the concept of OC cannot be objectively defined. He does however, suggest that OC has characteristics that are commonly agreed upon by most scholars who have made a contribution in this regard. These characteristics are that OC is holistic, historically determined, related to anthropological concepts, socially constructed, soft and difficult to change (Hofstede *et al.*, 1990). Hofstede extends this list of characteristics by also suggesting that OC is also manifested through practices that are acquired through socialisation at the workplace (Hofstede & Hofstede, 2001; Minkov & Hofstede, 2011).

In order to contextualise the influence of OC as a variable of any study of software development methodology, it is imperative that the abstractionism inherent in the variables underpinning such a study is reduced. Sekaran and Bougie (2010) posit that a common technique is to reduce the abstract notions to observable behaviour so that it can be quantified for the purpose of analysis. A viable strategy to reduce the abstractionism inherent in OC is to examine how this was achieved in the seminal publications by scholars such as Hofstede and Schein.

Hofstede's Theory of Organisational Culture

In an effort to understand the influence of OC on business processes, Hofstede and his colleagues conducted a study spanning 10 organisations and 20 organisational units (see Hofstede *et al.*, 1990). The research design involved the conducting of 180 interviews with top level managers coupled with 1300 survey responses that were received from employees at various levels of an organization. The survey that focused on the influence of OC on business processes consisted of 54 Likert scale type questions. A factor analysis of the responses to the 54 questions resulted in a 6-dimension classification of OC with each dimension containing a subset of the original 54 questions. The 6 dimension classification of OC proposed in Hofstede *et al.* (1990) is presented as opposing forces in an organisation. These allude to organisational behaviour that may be classified as process oriented or results oriented, employee oriented or job oriented, parochial or professional, open system or closed system, loose control or tight control and normative or pragmatic. While Hofstede's dimensions of OC provide a framework from which a viable attempt can be made to operationalise the amorphous concept of OC, it does not provide enough detail on how to achieve this transition. In order to obviate this shortcoming, a complementary perspective of organisational culture is obtained by examining the contributions made by Schein in this regard.

Schein's Theory of Organisational Culture

Edgar Schein, a professor at Massachusetts Institute of Technology (MIT) has established himself as a seminal author on the topic of OC. He has authored 15 books on management and OC. According to Lambrechts *et al.* (2011), Schein's contributions have been instrumental in shaping management practice and organisational scholarship.

According to Schein (1996, P. 32), OC can be analysed at 3 levels. These levels are listed in order from most to least superficial. At the most superficial level is 'Artefacts', a reference to the observed behaviour within an organisation. Schein does suggest that determining the type of culture that prevails within an organisation on the basis of analysing the observable behaviour in an organisation will in all probability produce an inaccurate interpretation of the prevalent culture. The 2nd level makes reference to 'espoused values and beliefs', primitively explained as 'the way things are done around here'. This aspect of OC evolves over a period of time where a specific problem solving strategy is critiqued, adjusted and transformed into an assumption that it will always be the correct way of solving a specific type of problem. While this is an indicator of how organisational values and beliefs are constructed and validated over a period of time (referred to as "social validation" (Schein, 1996, p. 26)), there is a low probability of empirically testing the link between performance and problem solving strategy. However, Schein makes the observation that when a solution to a problem works quite well, then the solution strategy that started off as a hypothesis is transformed into reality and becomes part of the set of underlying assumptions that defines processing within an organisation. This set of basic assumptions regarding reality, constitutes Schein's 3rd, and most substantive, level of organisational culture. At this level, Schein (P. 28) refers to a set of "dominant value orientations" that reflect the preferred solution among several alternatives, and members in an organisation will find it inconceivable to exhibit behaviour based on any other solution strategy. Iivari and Iivari (2011) named this behaviour as "enculturation" (p. 512), which refers to the process during which newcomers gradually learn by doing and observing how it is appropriate to talk, behave and act in an organisation. This enculturation activity takes place in order to establish a framework for stability and predictability within an organisation. Schein's explanation is that the human brain has a constant quest for cognitive stability and any challenge to the set of basic assumptions that preserve this stability will be met with a defensive response that seeks to uphold the existing cultural identity within an organisation.

The Need to Identify the Prevalent forms of Organisational Culture

This set of basic assumptions that defines the cultural identity within an organisation is regarded as one of the most significant factors that contribute to ASDM implementation failure (Chow & Cao, 2008; Howell *et al.*, 2010; Misra *et al.*, 2009). In order to ascertain the prospect of ASDM success within an organisation, the preceding discussion provides an a priori argument for focus to be bestowed on Schein's 3rd level of the OC framework. At the same time, the preceding discussion also identifies a need to establish how ASDM can be adjusted so that it aligns with the set of basic assumptions that defines the prevalent culture within an organisation. While there may be a temptation to look at the corollary arrangement whereby OC is manipulated and streamlined to suit ADSM, Iivari and Iivari (2011) point out that such an idea may not be feasible because we are referring to an "…anthropological and sociological phenomenon that is quite unique in every organisation" (p. 512). The preceding statement is quite significant in respect of the current study because the 'deep seated' nature of OC means that a prevalent strain of OC will not change in order to accommodate ASDM. However, given the flexibility inherent in ASDM, a logical suggestion is that ASDM could be customised to suit a specific strain of OC. The implication of this suggestion is that the prevalent forms of OC needs to be identified so that the adaptability of ASDM can be analysed with the objective of ensuring that there is organisational compatibility (Iivari & Iivari, 2011) with reference to these different forms of OC. The adoption of a software development methodology has to take cognisance of the prevailing OC. A failure to contextualise a software development methodology so that it has a resonance with the prevalent OC is one reason for the weak acceptance of software development methodologies in organisations (Iivari & Huisman, 2007).

2.6.1 Organisational Culture and Software Development Methodology

The relevance of OC to agile software development and software development methodology in general is highlighted in a seminal article by Alistair Cockburn and Jim Highsmith (see Cockburn and Highsmith (2001)) titled "Agile Software Development, the people factor". A verbatim comment made in this article reads as:

> *An agile team working within a rigid organization has as difficult a time as agile individuals working within a rigid team.* (P. 132)

Cockburn and Highsmith make the point that organisations that implement an agile approach to software development will not be successful if the de-facto command and control management style is maintained throughout the organisation. Organisations need to change norms and values to facilitate a leadership style that is collaborative rather than dictatorial. These norms and values (Hofstede & Hofstede, 2001; Minkov & Hofstede, 2011) as well as the basic assumptions (Gershon *et al.*, 2004) that are enshrined into OC have an influence on the software development process (Lee *et al.*, 2016) which is now considered to be a socio-technical process that incorporates organisational, human and technical components (Fuggetta & Di Nitto, 2014). A study by Claps *et al.* (2015) to determine the challenges faced by an organisation in adapting to a new software development methodology, it was observed that irrespective of the technical suitability of the new methodology, if it is not socially suitable, it will not be widely adopted. The study also found that the adoption of an agile oriented SDM required almost an organisation-wide commitment to enhance the prospect of successful adoption of the methodology. The role players ranged from senior managers to software developers to the end users of the system. The study adopted a technically-oriented theoretical model and there were many technical challenges that were identified. However, the researchers did acknowledge that it was the non-technical factor of organisational culture that was the biggest challenge.

From an academic perspective, the main challenge is to find appropriate academic theories that enable the study of technology from a human perspective. This challenge is somewhat alleviated by technology acceptance models such as the Technology Acceptance Model (TAM), Unified Theory and Use of Technology (UTAUT) and the Diffusion of Innovation Theory (DOI). As much as these models have enabled an incursion into the social realm with respect to technology usage and adoption, they do not provide optimal coverage for issues pertaining to organisational culture.

The difficulty of conducting OC research in the domain of software development methodology has been countered by researchers who have opted for the Cameron and Quinn (2011) Competing Values Framework (CVF) that explains OC using a quadrant-based orientation and provides a basis for the explanation of 2 dominant culture types in an organisation. These are the Stability and Control culture type and the Flexibility and Discretion culture type. In a study to determine aspects of OC that have an influence on the adoption of software process improvement (SPI) techniques, Lee *et al.* (2016) leveraged the CVF (explained in Section 2.6.2 and illustrated in Figure 2.14) to classify the types of organisational culture that are prevalent in an organisation. The outcome of this study was that the OC orientations that are aligned to flexibility and discretion promoted a more collaborative environment that was conducive to SPI initiatives. A plausible deduction is that the OC traits aligned to Flexibility and Discretion provided a fertile environment for the implementation of an agile approach to software development. The CVF framework was also used by Ngwenyama and Nielsen

(2003) who suggested that the lack of focus on OC in studies of software development methodology may result in a "blind-spot" error that may compromise any attempt to improve the software development process. In order to compensate for the "blind-spot" (p. 101) error, Ngwenyama and Nielsen (2003) used the CVF as the underlying theoretical model of OC for their study on software process improvement strategy in an organisational setting. A significant finding in this study was that organisations that strive for highly defined software development processes (as espoused by the Capability Maturity Model (CMM)) tend to adopt a cultural orientation that is lacking in flexibility and becomes more of an impediment towards the attainment of genuine software process improvement. This strategy of using the CVF was also used by Iivari and Iivari (2011) in their study of the relationship between organisational culture and the deployment of agile methods (explained in Section 2.6.2 and illustrated in Figure 2.14).

2.6.2 The Competing Values Framework (CVF)

According to Simberova (2015), amongst the numerous models that espouse to capture the essence of OC, the CVF is the most widely used and most widely cited. The CVF is a framework that is broadly informed by Hofstede's six-dimensional framework for OC. However, upon closer scrutiny, it is evident that the CVF is driven by the espoused values prevalent within an organisation and can be interpreted as a framework constructed on the basis of an overlap between Schein's (1996) 2nd and 3rd levels of organisational culture framework. As is the case with the classification of OC that was made by Hofstede and Schein, the CVF is made up of individuals with competing values and these values define the culture of an organisation (Quinn & McGrath, 1985). The CVF is based on the 2 dimensions of 'change versus stability' and 'internal versus external' forces, both of which provide a basis for the explanation of organisational behaviour (Iivari & Iivari, 2011; Simberova, 2015). These dimensions are reflected in the competing values of traditionalists and the advocates of innovation. They also form a subset of the 6 dimensional classification of OC proposed by Hofstede *et al.* (1990). The change/stability phenomenon emanates from Hofstede's dimension of pragmatic versus normative behaviour and the internal/external phenomenon of the CVF emanates from Hofstede's open system versus closed system orientation. The full CVF is based on an amalgamation of theoretical constructs underpinning OC that were proposed by 2 of the leading authorities on OC, Edgar Schein and Gert Hofstede. From an ASDM perspective, the CVF has significant relevance and has been widely used in information systems research in general (Iivari & Iivari, 2011). The original CVF was subjected to an adaptation by Denison and Spreitzer (1991) who performed a juxtaposition of the 2 dimensions of culture from the CVF that resulted in the emergence of four types of cultural orientations that define organisational behaviour. These are:

- Rational Culture – achievement oriented, where the focus is on productivity, optimisation of processes, accountability; internally, the focus is on economic use of resources and the external focus is on goal achievement and the attainment of competitive advantage;
- Hierarchical Culture – the focus is on stability and internal control with the underlying operational demeanour of ensuring security, control and stability by enforcing regulations prescribed by management structures;
- Group Culture – the emphasis is on flexibility and internal control; this aspect is strongly driven by the influence of staff members who use their individual and collective expertise and experience to determine the operational demeanour of an organisation; there is a strong focus on internal control, an aspect that profiles this type of culture as somewhat of a contrast to Developmental culture;
- Developmental Culture – the emphasis is on flexibility, and external focus; a direct contrast to group and hierarchical culture. Organisations that are anchored in this quadrant take risks, focus broadly about the big picture and big ideas, and are agile in their actions and the resources that they cultivate (De Graff, 2007); the initial investment in resources is mitigated by expectations of long terms benefits.

The Quadrant-Like Structure of the CVF

According to Denison and Spreitzer (1991), each of the culture types has its 'polar opposites', thus graphically manifesting as a 4 quadrant rectangular structure. On the basis of their study to investigate the relationship between ASDM and organisational culture, Iivari and Iivari (2011) extended this graphical manifestation of the CVF by identifying the main quadrant of applicability for ASDM as illustrated in Figure 2.14.

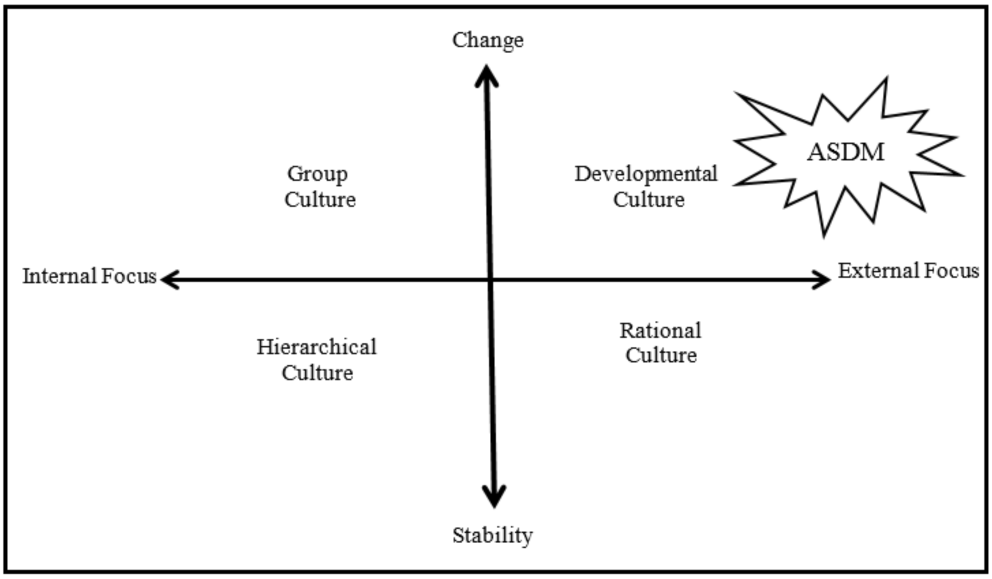


Figure 2.14: An Agile "Sweet-spot" in the CVF identified in Iivari and Iivari (2011)

The quadrant based classification provided by the CVF (illustrated in Figure 2.14) is further conflated into a classification centered on change and stability as well as an internal and external focus. The quadrants in the upper half of Figure 2.14 represent an alignment with change, flexibility and spontaneity, whereas the lower half of Figure 2.14 represent an alignment with strong control, continuity and order. Internal focus is a reference to the maintenance and preservation of existing/traditional socio-technical systems and culture within the organisation and is represented by the left half of the CVF in Figure 2.14. External focus emphasizes sensitivity to environmental issues where there is a focus on competition and interaction with the business domain elements that exist outside the organisation.

The Optimal Placement for ASDM in the CVF

According to Iivari and Iivari (2011), the optimal placement for ASDM is in the quadrant that represents a strong alignment with change and an external focus. As illustrated in Figure 2.14, the quadrant of optimal applicability for ASDM is situated in the upper right quadrant of the CVF model and has been named the Developmental Culture quadrant. According to Iivari and Iivari (2011), an organisational culture that espouses change and has a strong external focus is regarded as a 'fertile environment' wherein the principles of agility may be upheld, enabling the implementation of ASDM for the development of software systems.

While the CVF may be presented as a quantifiable, structured and theoretical model of organisational culture, Denison and Spreitzer (1991) warn that such a classification would rarely be found in reality. Although an organisation may be given an overall classification according to the CVF, by virtue of the presence of a single dominant type of culture, there is usually a presence of a mix of culture types that resonate between the various quadrants of the CVF. Iivari and Iivari (2011) also defend their contribution regarding the quadrant of applicability for ASDM, by asserting that while a Developmental Culture would be ideal for the deployment of ASDM, the methods underpinning ASDM have elements of the other 3 cultural types as well. As an example, features such as time-boxing, effort estimation and productivity (prominently used as part of Scrum and also relevant to XP), reflect values of a Rational Culture. Also, the Agile Manifesto is centered on behaviourist elements such as trust, motivation and commitment, all of which are traits of Group Culture. However, an argument for the compatibility between ASDM and Hierarchical Culture is not easy to defend and as such, it may be regarded as the least appropriate for the deployment of ASDM.

As suggested by Simberova (2015), the CVF has become a well-recognised standard for the classification of the type of culture that exists within an organisation. It is envisaged that the CVF will provide an ideal context for discourse on the influence of OC on the adoption and adaptation of ASDM. Also, Schein's concept of 'enculturation' provides an avenue whereby 'the way things are done around here' may be adjusted on the basis of success stories regarding the use of agile methodology. The main outcome from the discourse on OC is that the CVF provides a conduit from which a researcher may be able to understand the prevalent culture in an organisation. This knowledge will be crucial in identifying a software development methodology that is strongly aligned to the culture prevalent in an organisation.

2.7 Conclusion

The study's review of the literature was conducted using a 'funnel' approach that started off with a broad review of the software process improvement initiatives that have had a defining influence on the trajectory followed by software development process models. The literature review converged to a focus on iterative and incremental software development process models where there has been a unanimous endorsement by the professional software development fraternity for an agile approach to software development. A further specification in this regard is an overwhelming preference for the Scrum methodology because of the potential for the methodology to enable visibility of the evolving system thereby enhancing the prospect of generating quick business value. Scrum has also been endorsed because of its flexibility to handle changing user requirements. The adoption of Scrum methodology is however, susceptible to setbacks from a social and technical perspective.

A timeline illustration of the transition of the software methodologies (SDM's), as discussed in the content of the literature review, extending from 1940 until 2018 is represented in Table 2.5.

Table 2.5: Timeline Representation of SDM's

Timeline/ Dominant type of computer	1940 - 1960	1960 -1970	1970 -1980	1980 -1990	1990 -2000	2000 -2010	2000 - 2018	2010-2018
Mainframe	Code and Fix							
Mainframe		Structured Design						
Midrange/ microcomputer			Waterfall					
Personal Computer (PC)				Iterative and Incremental				
PC/Client Server					Spiral			
PC/Client Server						The Unified and Agile Unified Process		
PC/Client Server/Cloud Computing							Agile	
PC/Client Server/Cloud Computing								Wagile

The major advantages and disadvantages of each SDM is presented in Table 2.6 as SDM "sweet and bitter spots".

Table 2.6: SDM "Sweet and Bitter Spots"

SDM	Sweet Spot	Bitter Spot
Code and Fix	Quite simple; Gets the job done; Effective for simple/trivial systems; Enhances software maintainability through a "hacker" mentality	Lack of design compromised the stability of the system; Lack of formal requirements elicitation phase resulted in a system that did not meet user requirements; lack of testing compromised system reliability
Structured Design	High Level Design consisting of a hierarchy of sub-routines; enabling quick visibility; reduces complexity; Extensive use of Flowcharting	Rather vague in specifying the details of each level of the hierarchy; once design is in place, the flexibility is limited

Waterfall	Full support for entire development lifecycle; simple to implement; documentation intensive enabling greater accountability and maintenance	Too predictive; Users did not understand the system's requirements at the early; does not encourage an iterative demeanour because of it "single pass" nature; not much flexibility
Iterative and Incremental	Iteratively refine user requirements; handles changing requirements better than previous methodologies; rapid application development facilitate better system visibility	Not regarded as a true development process/methodology; may degenerate into a code and fix methodology; potential to produce "spaghetti code"
Spiral	Greater methodological presence; incorporated risk mitigation;	A complex methodology regarded as a "high ceremony" approach; substantial focus on documentation and reviews; system visibility is compromised; the model is theoretically sound but lacking in practicality and flexibility
The Unified Process (UP)	A truly adaptive methodology; extensive focus on upfront modelling and documentation; several iterations enabled better flexibility	Reliance on comprehensive upfront design and documentation reduced the capacity to faciltate quick system visibility thereby compromising the handling of changing user requirements
The Agile Unified Process	Greater focus on implementation and system visibility; better interactivity with end users; heralded a migration from technical aspects of development to the social context	Too process oriented compromising the flexibility somewhat
Agile	Embodies a fully dynamic and flexible process that is focused on the accommodation of user requirements and changing system specifications; simple design and short iteration cycles	Does not scale well; does not handle "mission critical" system development; lack of focus on design compromises the system's stability; Not practical to have an on-site end user; system's maintainability is compromised
Wagile/ Water- Scrum-Fall	Compromises dynamism to enable control and project management of the development process; incorporates greater focus on design; better control of "scope creep"; Incorporates adequate flexibility to let the development team decide on the level of control and agility	Suspicion of too much Waterfall focus thereby compromising the flexibility; tries to incorporate the best of Waterfall and Agile methodology thereby compromising each methodology individually

Many of the minor technical and logical impediments of SDM’s (referenced as “bitter spots” in Table 2.6) have been resolved by practitioners who have customised the methodology by intuitively making use of waterfall-oriented practices such as a BDUF and XP-oriented practices such as TDD, pair programming and code refactoring.

The major *technical* impediment manifests in the ability of the methodology to scale to an organisational platform. A consequence of this impediment has been the advent of organisation-wide variants of Scrum methodology such as DAD, DevOps and SAFe all of which have introduced the operations process as an integral part of the software development activity. There is however, still a lack of clarity and direction in terms of detail with regards to the integration of Scrum methods with operations process. The *social* impediment manifests in the form of OC and its influence on the adoption of an agile methodology such as Scrum. An inquiry to ascertain the influence of OC is difficult to accomplish because of its amorphous nature. The CVF does however, provide an operational guide to dichotomise OC so that it can be classified according to its scope of applicability to agile methodology. In order to guide the adoption of agile methodology, the CVF provide a classification structure that can be used as a basis to identify cultures within an organisation that resonates with the different variants of agile methodology.

The literature review has identified Scrum as the de facto agile methodology of choice for agile software development. However, the adoption of Scrum has been subjected to setbacks with regards to its scalability and its alignment with the prevailing culture in an organisation. This socio-technical area of Scrum adoption becomes a viable area for an inquiry that will enhance the implementation of the methodology thereby ensuring that the methodology achieves the intended objective of satisfying the customer “… through early and continuous delivery of valuable software” (1st principle of the Agile Manifesto taken from Beck *et al.* (2001)).

3.0 THE STUDY'S OVERALL DESIGN

3.1 Introduction

The literary incursion into trends and practices with regards to software development methodology has been concluded. It now becomes incumbent upon the researcher to provide a narrative with regards to the design of the study currently being undertaken. It should be noted however, that the literary analysis of the domain of software development methodology has provided much guidance in terms of the research design that needs to be followed for the current study.

A research design has to be aligned to the philosophical assumptions and the preferred paradigm that the researcher has adopted for the study, Creswell (2013). The abstractionism inherent in the preceding statement maybe somewhat obviated by the explanation from Scotland (2012) that a paradigm is a set of assumptions that a researcher makes about the researcher's interpretation of reality, also referred to as the researcher's worldview[18]. These assumptions are qualified by the structure provided in the philosophical concepts of ontology (a researcher's knowledge and interpretation of reality) and epistemology (a researcher's viewpoint of how new knowledge can be created). Pinch (2008) elucidates this incursion into the philosophical realm by suggesting that a researcher establishes a paradigm for research by adopting a research design that adequately addresses the duality between human and non-human phenomena that prevail in society. In order to address this duality in the context of the current study, the narrative that follows provides clarity on what has been achieved thus far, what needs to be accomplished and how this will be accomplished. In essence, the epistemology and ontology of the current discourse is substantiated with reference to the research questions that underpin the study.

One of the outcomes that has been achieved is the knowledge that Agile Software Development Methodology (ASDM) has been established as the current de-facto standard for software development methodology (SDM). At a more

[18] Morgan (2007) postulates that a worldview represents an all-encompassing disposition towards experiencing and thinking about world issues, including beliefs about morals, values and aesthetics.

granular level, Scrum has become entrenched as the most influential methodology that underpins software development. The current study may be seen as a contribution to the evolutionary trajectory that has been adopted in the implementation of SDM practice by engaging software practitioners in order to obtain a 'depth-driven' perspective on SDM's, with a specific focus on the methods espoused by ASDM. This foray into the experiential domain of software development by practitioners will be conducted predominantly from a technical perspective as well as a socio-technical perspective (as suggested in Pinch (2008)). The technical aspects will be aligned to the methods used in software development and the socio-technical aspect will have an exclusive focus on the influence that organizational culture has on the choice of SDM. In order to derive optimum value from an engagement with software practitioners, the overall research design adopted for the study is aligned to the Sequential Exploratory Design model suggested in Creswell (2013, p. 209).

The current chapter provides a philosophical basis for the choice of research design as well as a discourse on the methodology that will be used for the main phase in the study's design.

3.2 A Worldview Orientation

Mehra (2002) as well as Creswell (2013, p. 49) make the observation that qualitative research is underpinned by philosophical assumptions that 'drive' the methodological aspects of a study. The philosophical assumptions that constitute a researcher's worldview[19] are not a random occurrence, but formulated by dominant trends in the domain of the research discipline or based on past research experience, or as Mehra suggests, "…what we believe in determines what we want to study" (p. 8). The significance of this discourse on the worldview orientation towards research methodology is that a researcher's basic set of assumptions/beliefs is pivotal in determining whether the researcher adopts a qualitative, quantitative or mixed methods approach to their research (Creswell,

[19] Morgan (2007) postulates that a worldview represents an all-encompassing disposition towards experiencing and thinking about world issues, including beliefs about morals, values and aesthetics.

2013, p. 35). The dilemma in this regard is that it is difficult to add structure to a discussion of on amorphous concept such as a researcher's beliefs or philosophical orientation towards 'worldly issues'. Creswell does however resolve this dilemma somewhat, by reducing worldly issues to a human and non-human duality that is best understood according to 3 dominant worldviews, discussed in the discourse that follows.

3.2.1 The Post-Positivist Worldview

The post-positivist worldview is representative of the traditional form of research, also referred to as the scientific method and is based on observation and measurement of the objective reality that exists in the world. The post-positivist stance is an adaptation of the positivist worldview which advocates the belief that there is an irrefutable truth to scientific knowledge that consists of generalisations that are time-and-context-free (Guba & Lincoln, 1994). Based on the contributions by Kuhn (1970), the softening of the positivist stance on scientific knowledge resulted in a post-positivist worldview that embraced the importance of the context of scientific knowledge as well as the acknowledgement that the evidence provided by scientific research is not perfect and should be viewed as a conjectural truth rather than the absolute truth. The post-positivist worldview is intrinsically coupled with the quantitative approach to research. The researcher uses deductive reasoning and starts off with a research hypothesis, collects relevant data and tests the hypothesis with this data. A rejection or acceptance of the null hypothesis becomes the basis for the generation of a theory that may be subjected to further testing using new data and possibly in a different context.

3.2.2 The Constructivist/ Interpretivist Worldview

A worldview that is human-centric and embraces a philosophy that humans construct their own meaning thereby acknowledging the existence of multiple realities (Guba & Lincoln, 1994) that are subjective and based on interpretation. The research practice entails a quest by the researcher to unearth a complexity of views rather than aggregate viewpoints into one or a few generalisations. In contrast to the positivist stance, the truth is context-sensitive and there may be varied interpretations of an observation. According to Guba and Lincoln (1994), the research follows an evolutionary path that is highly inductive, where the focus is on theory generation rather than theory testing. Walsham (1993) does however issue the warning that theories do not represent the 'absolute' truth. The theories that emanate from a study underpinned by interpretivist philosophy should be judged according to the relevance and the 'excitement' that the theory generates in its domain of applicability. The objective of such research endeavour is to elicit in-depth, meaningful data from the phenomenon of interest.

The dominant methodology coupled with the interpretivist worldview is qualitative, where researchers ask open-ended questions in order to provide subjects with an open forum to express themselves with regards to their experiences and interpretations of world phenomena. The researcher's intent is to make sense of these interpretations and inductively generate a theory or a pattern that describes the collective experiences that subjects of a study may have in the context of the research phenomenon.

3.2.3 The Pragmatic Worldview

The pragmatic worldview embraces methods of both the post-positivist and constructivist worldviews. According to Creswell (2013), researchers make liberal use of methods that have underlying assumptions that are both qualitative and quantitative, in essence embodying a dualistic epistemology in the discovery of new knowledge. Pragmatism provides researchers with the freedom to choose quantitative and qualitative methods and procedures of research that best meet the objectives of the study. According to Petersen and Gencel (2013), the pragmatists viewpoint is to engage with research methodology on the basis of "…what is practically useful and whatever works at the time" (p. 2). It relies on a version of abductive reasoning (Morgan, 2007) that oscillates between induction and deduction. Morgan uses the term pragmatic reasoning to suggest that researchers who work in the pragmatic worldview use inductive results that have been obtained from a qualitative study to serve as inputs to meet the deductive objectives of a quantitative study and vice-versa. This strategy would be useful to address the dichotomy between the objective (positivist) and subjective (constructivist) research paradigms.

Morgan (2007) offers a summation of the discourse on worldviews by suggesting that:

- Qualitative research methodology conforms to an epistemology that is inductive, subjective and context bound;
- Quantitative research methodology conforms to an epistemology that is deductive, objective and generalizable;
- Mixed methods/pragmatic research methodology conforms to an epistemology that is abductive, intersubjective and transferable to different contexts.

3.2.4 The Software Engineering (SE) Worldview

Software engineering has its roots in computer science, thus placing it firmly in the realm of the reductive and deterministic domain of science where the dominant epistemology is positivism or post-positivism (Penny, 1997). Aligned to this traditional perspective is the commonly held perception that software development is an engineering-like activity (Pressman, 2010; Schach, 2008; Sommerville, 2007) and software developers are engineers who are provided with 'construction specifications' and are expected to proceed in a quantifiable, highly structured and organised manner to produce a software artefact that satisfies the pre-defined specifications (Jemielniak, 2008). However, this notion of developing software in a mechanistic manner has been subjected to a critique by Bryant is his philosophical foray into the origins and path trajectory of software development practice (see Bryant (2000)). In the article titled, "It's Engineering Jim ... but not as we know it", Bryant acknowledges that software development needs an exemplar discipline that embodies rigour, precision and quality, thereby endorsing an identity with the domains of engineering, mathematics and science. However, to suggest that the software development process should proceed in the same manner as the construction of an engineering-like artefact is unwarranted because the perception created is that the software development process embodies a methodology that is highly prescriptive and mechanical. Software development is also not governed by the physical laws of nature (as is the case with engineering), thus rendering the reference to engineering to be one that is more metaphorical than literal.

Engineering as a Metaphor

The metaphorical reference to engineering has served the domain of software development reasonably well in the sense that it has instilled an appreciation for the application of technical expertise, discipline and rigour into the development process. There were however, voices of dissent with regards to the appropriateness of the engineering/construction metaphor. The perception is that the engineering metaphor has outlived its usefulness and needs to be replaced by an image that portrays software development as an evolutionary process (Lehman & Ramil, 2003) where a software artefact is grown or 'nurtured' into a final product (Bryant, 2000). The 'softening' of the engineering metaphor has been necessitated by its incapacity to deal with the human/social element that has become a core component of the systems development process and the functionality offered by modern software systems. The understanding of software engineering has been coupled with a gravitation of opinion that was historically dominated by a scientific theoretical base to one that is more inclusive of the human and social aspects of computing. This movement espouses a migration from a worldview that was dominantly modernistic (the scientific perspective that the world could be explained by a set of rational and objective facts) to one that has been labelled as post-modernism.

Robinson *et al.* (1998, p. 368) explain that modernism "…lays the world bare, stripped of myth and mystery" and focuses on the rigid rules of determinism and the mechanistic logic of rationality that has an exclusive scientific orientation. This modernistic stance towards software development has created a mismatch between the actual process of software development and the type of methodologies that were advocated by the software engineering community to enhance the software development process. Robinson *et al.* suggest that adherence to the values of modernism by the software engineering community was a critical factor in causing the software crisis in the 1970's and 1980's creating a situation where the software engineering community became victims as well as perpetrators of the crisis. The complicity of the SE community in contributing to the software crisis is explained in Robinson *et al.* (1998) as a consequence of devaluing the user's experience and knowledge of their world, thereby creating a disconnect and a sense of inequality between the developer and the end user of a software system. In the traditional, modernist viewpoint, the software engineer is conceived as the expert whilst the consumer of the product is referred to as the end user of the system. The expert/end user distinction was suggestive of an arrangement where the expert made the decisions regarding the feature-set, performance and usability of a software system whilst the users was forced to adjust to the workings of the system that was bestowed upon them. This narrative is descriptive of an inflexible arrangement where the experts/developers were not obliged to provide any form of accountability to the end users of the system. It was at this juncture that the SE community began to acknowledge the need to have an interactive relationship with the end users in order to develop systems that were perceived to be successful.

The Importance of the End User

The acknowledgement of the importance of the end users role in the software development process has provided impetus for the popularity of agile methodologies, which according to Northover *et al.* (2007) heralds a paradigm shift in software development, completely replacing the old, traditional methodologies. From a philosophical perspective, this transition may be seen as a migration of the SE perspective on research and development from one of modernism to one that subscribes to the principles of postmodernism. Postmodernism is a philosophy that was developed as a rebellion against the positivist stance that reality could be explained using objective, rational thought and there was no flexibility to accommodate alternate explanations. Bertens (1995) explains that it is not easy to develop a precise definition of postmodernism and no single definition of postmodernism has gone uncontested or has even been widely accepted. Robinson and Sharp (2009) do however, provide some guidance and suggest that postmodernism is representative of a descriptive theory where there is no dominant narrative, enabling one to make an argument for knowledge that is constructed on the basis of the context in which the knowledge is discovered. Postmodernism is neither predictive nor prescriptive, but a philosophy that endorses a multiplicity of interpretations of factual knowledge. Software development practice typically consists of human intervention that embraces changing requirements and the ability of software developers to adapt software systems to suppprt the dynamic operational expectations of end users of these systems. These attributes are completely different to the modernisitic perspective that endorses engineering like precision where the physical artefact is predefined, foreseen and precisely built.

In his 1974 Turing Award acceptance speech (see Knuth (2007)), Knuth suggested that the activities of software development and research of the software development process were best understood when there is an acknowledgement that these activities embodied elements of modernism and postmodernism. Knuth qualified this claim by suggesting that the scientific approach (characterised by words such as logical, impersonal and calm) as well as the artistic approach (characterised by words such as aesthetic, creative, anxious and irrational) were intrinsic to the domain of software development. These sentiments heralded an acknowledgement that software engineering has an identity that is commensurate with the philosophy of postmodernism. The process of software development has to incorporate an amalgamation of modernistic traits such as logic and rationality, prescription and precision together with post-modernistic traits such as creativity and an appreciation of aesthetic quality. These predictive sentiments were empirically affirmed to some extent in a qualitative study by Jemielniak (2008). An interesting outcome of the study was that the dominant vocabulary and metaphors used by the software engineers had a stronger resonance with art rather than engineering. The resonance between software development and art has added impetus to the claim that an optimal understanding of issues related to software development is only acquired if there is sufficient cognisance accorded to the philosophy of postmodernism.

3.2.5 The Researcher's Worldview

In the context of the current study, the researcher makes a commitment to a philosophical stance that software engineering research has to be conducted in a constructivist/interpretivist space that does not preclude the use of positivist oriented methodology for software engineering research. The researcher's dominant worldview is essentially *interpretivist* with elements of *pragmatism*. Ontologically speaking, this viewpoint is classified as one of relativism, where knowledge is viewed as a social reality and it comes to fruition by virtue of the human context that is present in the discovery of knowledge. The researcher's interpretivist/pragmatic worldview orientation is not based on a *laissez-faire* attitude towards the worldview issue regarding software engineering. It has been carefully crafted on the basis of the researcher's personal experiences in the field of software engineering and also informed by the discourse on the dominant worldviews in the domain of software engineering as expressed by Bryant (2000); Dybâ and Dingsoyr (2009); Knuth (2007); Petersen and Gencel (2013) and Robinson *et al.* (1998).

According to Morgan (2007), there has to be an alignment between a researcher's ontological and epistemological perspectives and the choice of research methodology as well as the methods used to conduct research. Petersen and Gencel (2013) elaborates on this alignment in an article that deals with research methods and their relationship with a researcher's worldview in the domain of software engineering. In this article, a clear line is drawn between the interpretivist/pragmatist worldview, the extrapolation to qualitative research and the use of interviews or case studies as the main research method. In conclusion, the researcher's dominant worldview orientation will underpin the methodology and methods used in the current study, which at this stage has a strong resonance with the sentiments expressed by Petersen and Gencel (2013).

3.3 The Research Classification and Design Considerations

The research design is the grand plan that illustrates the "methodological congruence" (Morse & Richards, 2002, p. 34) between the research problem the research, the research methods and the collection and handling of research data. The most significant determinant of the research design adopted for a study is the researcher's worldview orientation (Saunders, 2011).

In the context of the current study, a narrative of the researcher's worldview orientation has been presented in the preceding section. It has been established that the researcher has a preference for the interpretivist worldview but in a more pragmatic sense. As Saunders (2011, p. 149) explains, "pragmatism is an intuitively appealing recourse" because it provides the researcher with the latitude required to enable focus on the research questions and the execution of the research process in a manner that is deemed to be feasible and doable. Before a commitment to a specific research design is proposed, it is essential to advocate a classification for a research project so that the research design can be identified in a manner that facilitates methodological congruence. According to Saunders (2011, p. 170) the three main types of research classifications are exploratory, descriptive and explanatory.

Exploratory research is a means of establishing 'what is happening' and to obtain new insight into a phenomenon. It is often used when knowledge of the research domain is vague and there is a deficiency of previous empirical research into the topic. The principal mechanisms of conducting an exploratory study is to compile an extensive literature review and to conduct interviews with experts on the subject matter. A descriptive study is undertaken in order to compile in-depth information about a person or event. Descriptive studies are rather static and are usually used as a precursor to an exploratory or explanatory study. An explanatory study is used when the main objective of the study is to establish a causal relationship between variables of a study.

In order to propose a classification type for the current study, reference is made to the first 2 research questions that underpin the study. These are:

- *What are South African software practitioners' perspectives on Agile Software Development Methodology (ASDM) from a technical perspective?*
- *How does organisational culture influence the implementation of ASDM?*

Both these questions require an in-depth engagement with software developers in South Africa. From an empirical perspective, there is a paucity of knowledge that is available from in-depth research oriented interactions with software practitioners in South Africa. Based on the argument presented, the exploratory approach is deemed to be the most appropriate classification for the initial empirical phase of the current study. The output of this phase of the study will consist of a static and a dynamic component. The static component will be a report on the current status of agile software development as experienced by the cohort of South African software practitioners. The dynamic component will be the development of a framework or a set of models that guide the future practice of ASDM. The choice of research design for the planned phases of the study is guided by the array of research designs presented in Creswell (2013, p. 11). An overview of these designs are presented for reference.

Quantitative Research Design

The two main methods of quantitative research designs are experiments and surveys. The underlying strategy is to generate numerical data that is context independent. The analysis procedures are predominantly deductive and entail the use of graphical analysis and statistical tests of significance to faciltate objective accuracy and enable generalization of results, Quantitative research designs have a strong affiliation to the positivist worldview that upholds the principles of objectivity and a single reality with minimal latitude for researcher bias.

Qualitative Research Design

Qualitative research is based on the philosophy that reality or knowledge can only be obtained in the context it exists. Qualitative designs rely on textual, image and video data that is analysed by the use of inductive and deductive methods to develop greater insight into a phenomenon. There is a strong alignment to the interpretivist worldview although Huberman *et al.* (2013, p. 7) suggest that it is more appropriate to label qualitative researchers as pragmatic realists. The purpose of qualitative research is to make sense of the complexities that exist around social phenomena from a cognitive and practical perspective. The participants in a qualitative study are selected on the basis of their knowledge or experience with the main phenomenon of the study. The outcome of qualitative inquiry is to discover new meanings, themes and generate explanations and conceptual frameworks to explain complex situations or cultures(Rubin, 2012).

Mixed Methods Research Design

Methodological pluralism is a strategy that is gaining traction in the domain of business and organizational research (Saunders, 2011) as well as research in the domain of information systems development (Frank *et al.*, 2014; Mingers, 2001). Traditionally, research in these areas have been dominated by a positivist philosophy where reality is considered to be objective and quantifiable. However, Mingers (2001), Petter and Gallivan (2004) and Frank *et al.* (2014) point out that the multi-disciplinary nature of studies within information systems development necessitates multi-method research approaches that embrace both positivist and non-positivist traditions. This strategy will help to broaden understanding because it incorporates elements of objectivity and scientific rigour as well as interpretivist and qualitative rigour. The diversity of research approaches will arguably ensure that the researcher is in a position to leverage the strengths of both research paradigms thereby mitigating the complexities inherent in information systems development research.

According to Saunders (2011, p. 185), mixed methods research is a reference to a research design where quantitative and qualitative data collection and analysis techniques are used for different phases of a study. The phases of such a study are done either concurrently or sequentially. The choice of design strategy is guided by the researcher's intuition and perspective.

3.4 The Research Design

The possible research designs that could be implemented in the current study have been presented in the previous section. This narrative should facilitate a choice of research design quite easily. However as Saunders (2011, p. 185) points out, the choice of design is still not easy to identify. In the context of the current study, the researcher has made a definite commitment to the interpretivist/constructivist/pragmatic worldview in the discourse on the worldview perspective. This commitment suggests a leaning towards a more qualitative oriented design, In order to validate this inclination, guidance is obtained from Mingers (2001) who provides a conceptual framework for information systems (IS) research design that is based on the philosophical works of Kuhn Kuhn (1970) and Habermas (1970). Mingers advocates a conceptual framework of research that is based on the construct of "three worlds". These are the:

- *Material world*: a world that is outside and independent of human beings and observations in this world are objective and theory-driven;
- *Personal world*: a world that consists of human beings own individual thoughts, feelings and experiences and observations in this world are subjective and constructed by experience;
- *Social world*: a world that represents the co-existence of human beings and is driven by inter-subjectivity that consists of social practices, norms and values that enable and constrain the action of human beings in a society.

Mingers (2001), Saunders (2011) as well as Creswell (2012) are of the opinion that research in general is not a discrete event and entails a set of phases that ask the questions: *What is happening* with reference to a specific phenomenon, *why it is happening* and *what can be done* to implement a change so that there is an improved experience of that phenomenon. Each of these questions may be answered using multiple research approaches. From an IS/IT perspective, the significant benefit of using a multi-method approach is that it will arguably ensure the synthesis of a reality that captures the technical and social complexities that define the IT/IS domain.

Having made a commitment to an overarching exploratory design for the current study as well as an endorsement of the value of a mixed methods research approach for IS/IT based research, the researcher enlisted the guidance provided in Creswell *et al.* (2003) on the topic of possible designs for a mixed methods study. The objective of this exercise was to identify a viable research design that enabled the answering of the research questions for the current study.

The main criterion that provides a distinction in terms of the type of mixed methods approach is the sequence that is used to collect and analyse data. The data collection for the multiple phases may be done concurrently or sequentially. The sequential strategy entails an initial phase that is either qualitative or quantitative and a subsequent phase that reverses the methodology. Based on these 3 core mixed methods designs, Creswell *et al.* (2003, p. 167) introduce 3 additional distinguishing criteria. These are the overall priority that is attached to the type of research, the point at which integration of data is conducted and the relevance of a theoretical framework to underpin the study. Aligned to this framework of research designs, the design for the current study is structured along the specifications.

- The design will have an overarching qualitative focus to enable an in-depth exploration of the phenomenon (of agile software development);
- The design will enable a convergence of ideas and experiences so that a holistic explanation of the phenomenon of software development in South Africa may be ventured;
- The design incorporates the synthesis of a set of models that captures the essence of the qualitative component of the study;

- The design provides an option whereby the researcher is able to validate the model by making use of a theoretical framework to ascertain acceptance of the proposed model using a quantitative approach. The choice of a quantitative approach in the latter phase of the study is guided by the suggestion in Saunders (2011, p. 185) that the use of different techniques in a complementary manner is sometimes advisable to cancel out the “method effect”. The “method effect” is a reference to the shortcomings that may be present in a single strategy and the cancelling out of this shortcoming is achieved by enlisting the service of a complementary approach. It is claimed in Saunders that this strategy will arguably ensure that there is greater confidence that may be placed in the conclusions of the study.

Using the specifications listed above, the most appropriate research design identified for the current study is the Sequential Exploratory Design suggested in Creswell *et al.* (2003, p. 180). In terms of the notation used to describe this type of research design, Morse (1991) suggested the use of the expression QUAL →quan to denote a study that has an overarching qualitative design but also makes use of a subsequent quantitative approach that is dependent on the outcome of the qualitative phase of the study. The phases of the study are executed in a sequential manner.

3.4.1 The Sequential Exploratory Research Design

The sequential exploratory design is conducted in two phases where priority is attached to the first phase of the study. The initial phase consists of a qualitative data collection and analysis phase followed by a quantitative data collection and analysis phase. The final phase entails an integration of the analysis from the both phases. The intention of the first phase of the study is to explore the problem under study and then follow up with a quantitative phase that seeks to obtain validation of the outcome obtained from the first phase of the study. The sequential exploratory design adopted for the current study is illustrated in Figure 3.1.

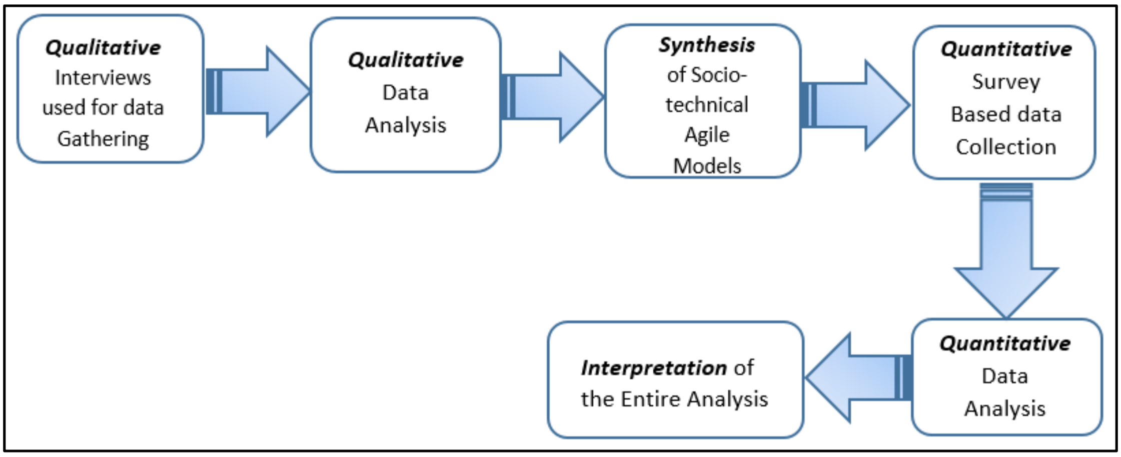

Qualitative Interviews used for data Gathering
Qualitative Data Analysis
Synthesis of Socio-technical Agile Models
Quantitative Survey Based data Collection
Quantitative Data Analysis
Interpretation of the Entire Analysis

Figure 3.1: The Sequential Exploratory Design Adapted from Creswell *et al.* (2003, p. 180)

In order to contextualise the use of the Sequential Exploratory Design illustrated in Figure 3.1, reference is made to the inception of the current study which was at the literature review phase. The literature review conducted in Chapter 2 entailed an elucidation of the practices and methods used in software development and was predominately technical. However, the relevance of organisational culture in understanding the technical issues became integrated with many of the technical issues underpinning software development. The culture within an organisation consisted of habits and practices that became embedded into software development techniques. This phenomenon necessitated a foray into the socio-technical domain of software development. An outcome of this process is the instantiation of the first 2 research questions[20] which necessitated an overarching qualitative[21] approach towards the study. The first 2 phases in the sequence of the research design (illustrated in Figure 3.1) is attributed to the first 2 research questions. It should be noted that the strategy of using a literature review to inform the research question(s) in a qualitative study is endorsed by Creswell (2013, p. 50) who asserts

[20] The first 2 research questions have been repeated in Section 3.3 for ease of reference

[21] A discourse on the choice of an overarching qualitative approach for the study is presented in Section 3.5

that sufficient flexibility may be accorded to the qualitative researcher, unlike in quantitative research where a theoretical framework is given much more prominence. The lack of reference to a specific theoretical underpinning renders this phase of the study as inductive. According to Thomas (2006) the main purpose of an inductive approach is to allow research findings to be obtained from the significant themes found in the raw data without the constraints imposed by any structural underpinning such as academic theory or a structured methodology. Thomas elaborates on the purpose of an inductive approach and suggests that the following outcomes are expected from the inductive approach:

- a condensation of extensive and diverse raw data into a cogent, summarised form;
- an alignment of the research objectives and the summarized findings that is transparent and defensible;
- enables the synthesis of a model or theory about the underlying structure of experiences or processes that are found in the data.

The synthesis phase in the current study forms the 3rd phase of the study (illustrated in Figure 3.1) and entails the development of a set of models that guide the implementation of agile software development methodology (ASDM). The models will have a social and a technical orientation. The social dimension will be aligned to organisational culture theory. It is envisaged that the organisational culture dimension of the study will provide an overarching framework to match the culture within an organisation to the type of development methodology best aligned to this culture. The technical dimension will be largely dictated by the evidence in the qualitative, exploratory phase of the study. This phase of the study will provide an answer to the 3rd research question, listed below for reference.

- *How can South African software practitioner's knowledge of ASDM be used to develop a framework to guide the implementation of ASDM?*

The 4th phase of the study follows a quantitative approach that entails an inquiry to determine the level of acceptance by software development practitioners of the technical component of the proposed framework for the implementation of ASDM. The researcher has made a conscious decision to validate the technological component of the proposed framework because of its envisaged pragmatic relevance to practitioners. This decision resonates with the suggestion by Creswell *et al.* (2003, p. 171) that practical constraints with regards to data collection coupled with the "amenability" of the research approach to the validation exercise will determine the level of intensity of the quantitative phase of the study. A social science-oriented theoretical model of technology acceptance will be used to operationalise software practitioners' acceptance of the proposed technological model to guide the implementation of ASDM. It should be noted that the design for the quantitative phase of the study is only tentative at this stage (as indicated in Creswell *et al.* (2003, p. 171)). It will however be used to answer the final research question, listed below for reference.

- *What is the acceptance by South African software practitioners of a framework that informs the technical implementation of ASDM?*

An analysis and condensation of the knowledge obtained from the 1st 5 phases of the study will lead to a conclusion of the study. This conclusion will entail the 'mixing' of results in order to present an element of sequential triangulation which is one of the hallmarks of a successful mixed methods study (Morse, 1991).

The remainder of the chapter comprises of a discussion of qualitative research methodology, the approach that underpins the first (and defining) phase of the study.

3.5 Qualitative Research Methodology

According to Leedy and Ormrod (2005, p. 133), qualitative research is a multi-faceted research approach that entails the study of "real world phenomena" together with all the complexities that define these phenomena. The complexities inherent in these phenomena make it impossible to simplify the outcome of a qualitative inquiry so that it converges to "…a single, ultimate truth to be discovered". The underlying philosophy of qualitative research is that the researcher should have the ability to interpret the results of a qualitative study by not only appreciating the immediate outcome of a study, but also to be cognisant of the broad social context in which the study has been undertaken. Remler and Van Ryzin (2011) suggest that qualitative research is best defined according to the kind of data that it generates as well as the methods used to analyse this data. The data is primarily nonnumeric consisting of textual data that provides an insight into the thoughts and experiences of the human subjects of the study. In order to present a solution to a research problem, the researcher uses the newly acquired insight into the problem to develop a theory to explain the problem or construct a model that represents a solution.

Qualitative research does have its limitations in the sense that the results may be not be generalizable to a broader population. The lack of generalisability is however, made up by the depth of understanding that a qualitative study produces, thereby providing a forum to enhance the understanding of issues that are humanistic and underpinned by a strong social context. Aligned to the preceding assertion is the claim by Remler and Van Ryzin (2011) that qualitative research is ideal for exploratory studies or studies that attempt to understand social and organisational behaviour that is deemed to be vague or not easily explained.

3.5.1 The Use of Theory in Qualitative Research

The theoretical framework forms the blueprint for the entire dissertation inquiry (Grant & Osanloo, 2014) or as is suggested in Sekaran and Bougie (2010), the theoretical framework provides the conceptual foundation from which the research project evolves. Sekaran and Bougie do make the claim that the theoretical framework forms the underpinning of the hypothetico-deductive research method because it informs the research hypotheses used to guide the research process. However, this claim has a strong bias towards quantitative research projects that are typically ‘driven’ by deductive logic. In these instances, the theoretical framework provides an indication of the researcher’s beliefs or theory of the relationship(s) between the variables of a study. This theory becomes the focus of a testing process where the objective is to use statistical methods to either support or reject the researcher's theoretical stance within the context of the study's data.

However, qualitative research is typically, not initiated with any dominant theoretical disposition, does not have an exclusive reliance on deductive logic and "…produces findings not arrived at by statistical procedures or other means of quantification" (Strauss & Corbin, 1998, pp. 10-11). The methodology of qualitative research facilitates the generation of a theory or a pattern that emerges inductively towards the end of such a study (Creswell, 2013). Creswell also points out that in some instances, qualitative studies may not include an explicit theory and entails a presentation of descriptive research of the central phenomenon.

3.5.2 A Choice of Qualitative Methodology

Creswell (2013, p. 13) and Glesne (2015, p. 20) provide a list of prominent qualitative research methodologies that include narrative studies, ethnography, phenomenology, grounded theory and the case study approach. These methodologies leverage a common set of methods that provide the researcher with an identifiable strategy to enable an understanding of the phenomenon that forms the center of the inquiry. From an overview perspective, Creswell provides a useful summary of the purpose of the main qualitative methodological approaches. From the perspective of the current study, the most applicable approaches were the following:

- Phenomenology: the researcher obtains an insight into the lived experiences of individuals about a phenomenon; this insight is used to develop a framework/picture or a cluster of themes that encapsulates the individual experiences and enables the understanding of the phenomenon in a broader context. The main form of data collection is an interview with the individual who has experienced the phenomenon;
- Grounded Theory: the researcher develops an abstract theory of a phenomenon that is based on knowledge of the phenomenon as

conveyed by the study's participants. Data collection is done iteratively until a convergent view is established;

- Case Study: the researcher develops an in-depth analysis of a case that may include an event, a process or activity or one or more individuals over a sustained period of time. A variety of data collection procedures may be employed including interviews, documentation, observational notes and surveys.

The choice of research approach for the current study is guided by the researcher's ontological perspective regarding research in the domain of software engineering (discussed in Section 3.2.5) as well as current research trends in the domain of software engineering. The use of Grounded Theory has been hailed by Stol *et al.* (2016) as a viable alternative to the tradition of following a hypothetico-deductive research model. However, a major source of concern regarding grounded theory research with regards to software engineering is that there is a high probability of method slurring, a situation where the researcher does not engage with the methodology in a rigorous manner. One of the reasons for this phenomenon is that theory generation in an amorphous discipline such as software engineering is difficult to achieve because of the number of variables that may be involved.

Case Study as a Viable Methodology

The use of a case study approach is also a viable methodology that could underpin the current study. According to Yin (1981), the case study approach entails an inquiry regarding a phenomenon in a specific context. While the preceding outcome may be deemed as partially appropriate for the current study, it was not sufficient to enable the acquisition of knowledge regarding the use of agile software development methodology (ASDM) from a broader context. Phenomenology enables the researcher to obtain a broad perspective on the topic as has been illustrated in studies of ASDM by Nguyen (2016), Matthews (2014), Malone (2014) and Mayfield (2010). This precedent of using phenomenology for software engineering research coupled with the perceived shortcomings of the grounded theory and case study approaches, makes phenomenology a viable qualitative methodology to underpin the current study.

3.5.3 Phenomenology as a Viable Qualitative Methodology

A phenomenological study is a qualitative discourse that "…describes the common meaning for several individuals of their lived experiences of a concept or a phenomenon," Creswell (2012, p. 76). Phenomenologists arrange their inquiry by first establishing a phenomenon of human interest and then proceed to obtain knowledge of that phenomenon by eliciting details of people's experience(s) by virtue of their interaction with the identified phenomenon. The underlying intention is to establish a noetic ("how did you experience the phenomenon?") and noematic ("what is the value that may be derived from your experience of the phenomenon?") correlation (Langdridge (2008); (Groenewald, 2004)).

Chan *et al.* (2013) further explains that phenomenological research is based on the ideology that a better understanding of a phenomenon is obtained by analysing the experiences of the phenomenon by the subjects of a study. The phenomenological strategy is to ask the interviewee an initial question that opens up a channel of communication to enable a deeper inquisition of the subject matter. The objective here is to acquire general knowledge that is based on the interviewee's experience and learned perspective of the phenomenon. In order to conduct phenomenological research, the researcher should however have some knowledge on the presence of the phenomenon as well as an intuitive list of respondents who will have sufficient experience in the phenomenon. Basically, the researcher should have an interest in the phenomenon and knowledge of the parameters that define the phenomenon.

The discourse on phenomenology, coupled with the researcher's epistemological viewpoint that reality is constructed by virtue of an individual's *subjective* experience of a phenomenon, has resulted in a plausible argument for the use of phenomenology in the current study. However, the main methodological aspects of phenomenology need to be established in order to ensure that the phenomenological inquiry is based on sound theoretical principles. From an operational perspective, Englander (2012) provides an insightful explanation of the methodological aspects of phenomenological research by contextualising the operational elements according to the expectations of logical positivism, which has assumed the role of the de facto methodology for research in the era of modernism. From a positivist perspective, the initial step of data collection is sampling, which emanates from the notion that the sample needs to be identified so that observations regarding the sample may be statistically generalisable to the population at large. The critical questions that drives this process is: *Is each element of the sample representative of each element in the population?* And, *how many elements are required in the sample so that observations can be statistically inferred onto the population?*

However, in phenomenological research, representativeness is not the main criterion that drives the methodology. While there is a quest for general knowledge of the phenomenon, this is acquired more from a 'depth' rather than a 'breadth' perspective. The main criterion that drives the identification of respondents for the study is the answer to question: *Do you have the experience that I'm looking for?* In terms of the sample size, Smith *et al.* (1997) is of the opinion that there is "…no right answer to the question of sample size" (p. 56) and unlike logical positivism, the sample size is determined by the richness of the evolving data collection process. Englander (2012), concurs with this assessment of the issue of sample size and makes the point that because the study is qualitative, the sample size does not really matter. There is however a 'veiled' agreement that the more interviews you conduct, so will your understanding of the phenomenon improve. In this regard, the actual sample size could be anything from 3 to 20 respondents. The maximum number suggested is based on the assumption that at some stage, there will be a convergence of the information gathered so that no new information becomes available in which case a point of data saturation has been reached. The main sampling strategy for qualitative research is purposive (Huberman *et al.*, 2013, p. 31) and these samples are not necessarily pre-specified and identification of potential respondents for the study can evolve during the course of data collection.

With regards to the data collection instrument, unlike logical positivism where the data collection instrument is seen as a device of *measurement*, in phenomenological research, the data collection instrument is a device used to elicit *meaning*. The research instrument represents an opportunity to become acquainted with the phenomenon via the interpretation of the person/interviewee, without being overly concerned about the individual/demographic traits of the person. As Chan *et al.* (2013) explains, the ultimate goal of phenomenological research is to gain an intimate understanding of the lived experience of the interviewee. An ideal strategy would be to make use of open-ended questions to elicit the experiential data as well as a semi-structured interviewing technique so that general knowledge regarding the phenomenon is obtained from the interviewee. The underlying strategy is to enhance the prospect that the interview questions are developed around the research aims.

With regards to the analysis of the interview data, the dictates of qualitative research analysis come to the fore. As suggested by Huberman *et al.* (2013, p. 14), "…qualitative data analysis is a continuous, iterative enterprise. Issues of data condensation, display, and conclusion drawing/verification come into play successively as analysis episodes follow each other" as illustrated in Figure 3.2. This process is conceptually similar to that followed by quantitative research where there is a preoccupation with data condensation via the calculation of means and standard deviations, data display via correlation tables and regression printouts and conclusion drawing via the reliance on significance levels and experiment/control group differences. However, in quantitative research, the activities are carried out in more of a sequential manner. In qualitative research, the transition between activities is more iterative (as illustrated in Figure 3.2).

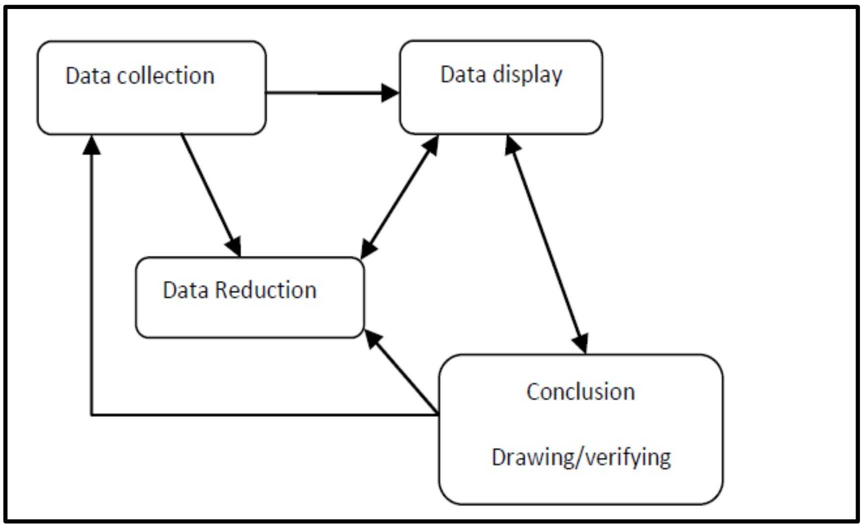


Figure 3.2: An Iterative Model of Qualitative Data Analysis (Huberman *et al.*, 2013, p. 14)

3.6 The Main Phenomenon of the Study

The current study's design may be perceived as an evolutionary one. Based on the knowledge gleaned from the literature review, it has been established that the agile methodology, specifically the scrum-oriented version of the methodology, seems to have been established as the de facto standard for software development. However, embedded in this knowledge is also the awareness that practitioners are using customised versions of agile methodology for software development. A significant imperative that follows is the attainment of knowledge with regards to the issues that underpin the customization of agile methodology from a South African perspective. The idea is to uncover the essence of the software craft knowledge (also referred to as "software crafting" in Boehm (2006, p. 13)) that prevails in South Africa so that this knowledge can be used to 'fuel' the development of a practitioner-informed, agile based software methodology guiding framework. One of the challenges associated with achieving the afore-mentioned imperative is to implement a research strategy that can be defended from a philosophical and methodological perspective. In order to achieve this, reference is made to a paper by Barry Boehm, titled "20th and 21st Century Software

Engineering”, where Boehm presents a discourse on the current trends as well as a prognosis for the direction of research and practice in the field of software engineering (see Boehm, 2006). Boehm structured the paper by using a strategy whereby the discourse was presented according to the dictates of the philosopher, Georg Hegel, who hypothesised that:

> *…increased human understanding follows a path of thesis (this is why things happen the way they do); antithesis (the thesis fails in some important ways; here is a better explanation); and synthesis (the antithesis rejected too much of the original thesis; here is a hybrid that captures the best of both while avoiding their defects).*

This Hegelian perspective of thesis, antithesis and synthesis provides an ideal philosophical framework that defines the current study. The assertion is corroborated by the study’s plan which in essence consist of a thesis (establish a trend with regards to the current practice of agile based software development projects in South Africa), antithesis (ascertain reasons for the customisation of agile based methods and elicit suggestions for an improvement to the agile methods) and a synthesis (propose a framework that is based on agile methodology that incorporates the suggestions from practitioners on how agile methodology can be improved within the South African context).

Hegel’s philosophical outlook, as explained in Stern (2002), is strongly aligned to phenomenology. According to Dowling (2007) and Kafle (2013), phenomenology is a term that has a dual context. It is regarded as a philosophy as well as research methodology. From a philosophical perspective, phenomenology (considered to be a branch of epistemology) has a focus on the cognition that occurs as people construct knowledge on the basis of reflection and experience of their “lifeworld” (Langdridge, 2008, p. 1128) . From a research methodology perspective, Leedy and Ormrod (2005, p. 139) define a phenomenological study as “…a study that attempts to understand people’s perceptions, perspectives and understandings of a particular situation”. One of the core objectives of the current study is to acquire an understanding of software practitioners’ perspectives on the phenomenon of software development. The preceding narrative, regarding the

Hegelian perspective on knowledge acquisition, the essence of phenomenology as a research methodology and the objective of knowledge acquisition regarding the phenomenon of software development by practitioners in South African, provides rational testimony to support a gravitation of the study's methodological underpinning towards phenomenology.

3.5.4 Main Types of Phenomenological Approaches

According to Chan *et al.* (2013), there are seven approaches to phenomenological research. However, the two main types of phenomenological approaches are descriptive and hermeneutic. In the case of descriptive phenomenology, the researcher employs a strategy named 'bracketing' where every effort is made to ensure that the researcher's experiences, knowledge and opinion on the topic of the study is not used to influence the interviewee's responses. Basically, the influence of the researcher has to be 'bracketed-away' enabling the interviewee to provide an organic response. This form of phenomenology has many critics (e.g. Chan *et al.*, 2013; LeVasseur, 2003) who claim that it is impossible for the researcher to eliminate pre-understanding of the topic and there has to be a point where this pre-understanding influences the conversation with the interviewee.

Much of the discourse on phenomenology is attributed to the philosophers, Husserl and Heidegger. Heidegger was of the opinion that the biases and assumptions of the researcher cannot be 'bracketed away' and is embedded within the interpretive process of engaging with the interviewee (Laverty, 2003). This interpretivist attitude towards a phenomenological study has its origins in the ontological perspective that there are multiple realities/interpretations that underpin the experience of a phenomenon and reality can only be appreciated in the local context in which it has been created and experienced. Reality is not necessarily a global phenomenon. The preceding statement also serves as a powerful endorsement of the qualitative/interpretivist paradigm of research.

Laverty (2003) advises that the researcher should be creative and adopt approaches that enable an optimal response to a question thereby enhancing the prospect of obtaining rich, meaningful insight into the interviewee's experience and perception of a phenomenon. In order to achieve this Englander (2012) suggests that the initial phases of an interview should adopt a bracketing approach and hereafter, the hermeneutic approach may be used to obtain deeper insight into the interviewee's experiences as well align this with the objectives of the study.

The research instrument used in the qualitative phase of the study (see Appendix A) is designed to incorporate elements of bracketing and hermeneutics.

3.6 Conclusion to the Research Design

At the outset, the research plan has been largely dictated by the researcher's worldview that gravitates towards an interpretivist, pragmatic orientation, resulting in the adoption of a predominantly qualitative approach to underpin the study. In order to mitigate for the potential influence of methodological bias, a quantitative phase was included in the study. From the perspective of research methodology theory, the study's design is aligned to the Sequential Exploratory design defined in Creswell *et al.* (2003), an embodiment of a mixed methods research approach. The qualitative phase of the study was conducted using a phenomenological approach. From an empirical perspective, data for the qualitative phase of the study was obtained through in-depth interviews conducted with software practitioners experienced in the domain of agile software development methodology and who have expert knowledge in the domain of general software development. The output of the exploratory phase is the development of a set of socio-technical models to inform the use of agile software development methodology. The technically oriented output from this phase is subjected to a quantitative validation process. The quantitative validation is underpinned by technology acceptance theory and implemented through a survey based approach for data collection. The qualitative and quantitative data analyses is conducted independent of each other. The study's conclusion is used as a platform to achieve sequential triangulation and explain the convergence the results.

A Synopsis of the Research Design

A synopsis of the research design is contextualised according to the research questions underpinning the study and presented in Table 3.1.

Table 3.1: Research Methodology Aligned to Research Questions

Research Question	Research Approach	Main Activity
What are South African software practitioners' perspectives on Agile Software Development Methodology (ASDM) from a technical perspective?	Qualitative	Interviews guided by **Phenomenological Theory**
How does organisational culture influence the implementation of ASDM?	Qualitative	Interviews Guided by the **Competing Values Framework**
How can South African software practitioners' knowledge of ASDM be used to develop a framework to guide the implementation of agile methodology?	Qualitative Synthesis Phase	Qualitative Analysis using the **Van Kaam method for qualitative data analysis** (explained on P. 188)
What is the acceptance by South African software practitioners of a framework that informs the technical implementation of ASDM?	Quantitative	Survey and quantitative analysis informed by the **Theory of Acceptance of Software Development Methodology** (TASDM)

The answers to the study's research questions provide a convergence to the study's main question that has been specified as:

How can experiential knowledge of Agile Software Development practice be used to develop a Socio-technical Framework to Guide the Implementation of Agile Software Development Methodology?

As can be established from Table 3.1 the study has been designed with a mix of qualitative and quantitative methodologies. The qualitative methodologies have been used in an exploratory manner and the quantitative methodology has been used in a confirmatory manner. The main research question alludes to the development of a socio-technical framework. This framework is developed on the basis of the experiential knowledge obtained from the cohort of software practitioners that form the sample for the qualitative phase of the study. A verification of the framework is then obtained by leveraging quantitative research methodology and the Theory of Acceptance of Software Development Methodology to survey a cohort of software practitioners on the viability of implementing the proposed framework.

4.0 THE QUALITATIVE DATA

4.1 Introduction

According to Creswell (2012, p. 19), methodology is a reference to the process followed in achieving the research objectives. An integral component of the methodology for the *qualitative phase* of the study is the process adopted to obtain the qualitative data. The objective of the current chapter is to provide an insight into the qualitative sampling approach and the interview protocol used in the study. The current chapter 'sets the scene' for the qualitative data analysis (Chapter 5) by including a discussion on:

- the qualitative sampling approach and the sample size;
- the design of the interview guide/questions for qualitative data collection;
- the pilot study;
- the attributes of the study's participants.

4.2 The Sampling Approach

The dominant sampling technique for qualitative research is purposeful sampling (Given, 2008; Leedy & Ormrod, 2005; Remler & Van Ryzin, 2011) because the research objective in a qualitative inquiry is to obtain an in-depth view of the main phenomenon of the study. In order to achieve this objective, the methodology has to enable the selection of an appropriate, information-rich sample that typically ranges from a single case to a relatively few cases that are purposefully selected (Patton, 1990). Patton qualifies the concept of 'information-rich' cases as those cases from which one can obtain maximum knowledge about issues that are of central importance to the purpose of the research. Patton (1990, p. 182) and Given (2008, p. 697) provide a listing and an explanation of the purposeful sampling techniques that may be used to obtain knowledge of information-rich cases. The underlying theme that emanates from this discourse on purposeful sampling techniques is that they are not mutually exclusive and the researcher is advised to make a selection that enhances the prospect of obtaining information-rich cases that enable an optimal illumination of the issues pertaining to the research question(s). Patton does however concede (p. 181) that there is no perfect choice of a purposeful sampling technique and the researcher should select a technique that fits the purpose of the study, the questions that are asked and the resources that are available.

The phenomenological approach adopted for the current study necessitated the selection of respondents based on the following criteria:

- The respondents must be software practitioners who have had at least 5 years of experience in software development in an organisational context;
- The respondents must have at least 2 years of experience in the use of agile software development methodology or in the use of methods that are intrinsic to agile software development.

The criteria identified for the selection of respondents is aligned to the purpose of the study. The purposeful sampling used is a mix of criterion-based sampling, snowball sampling and opportunistic sampling. These purposeful sampling techniques have been described in Patton (1990, p. 183) and Given (2008, p. 697) as:

- Criterion-based sampling: identifying information-rich cases that meet some criterion;
- Snowball/Chain sampling: Use the identified subjects of a study as a source to obtain knowledge of cases of interest of other people who meet the criteria for the study and to continue this process iteratively;
- Opportunistic sampling: The researcher makes ‘on the spot’ decisions to take advantage of new opportunities during data collection. This approach capacitates the researcher to identify new opportunities from which information-rich data may be generated.

The purposeful sampling strategies adopted in the current study consists of a hybrid of planned and unplanned data collection. The planned component (criterion-based) entails an identification of subjects who meet the set criteria for the study. Patton (1990) does however suggest that one of the strengths of purposeful sampling is the ability of the researcher to be agile and identify opportunities that may develop after fieldwork has begun. This approach permits the sample to emerge during the course of fieldwork and is aligned to the snowball and opportunistic sampling techniques.

4.2.1 Sample Size

With regards to sample size, many of the prominent authors of qualitative research methodology are of the opinion that there is a trade-off between breadth of the study and the depth of the study (Patton, 1990, p. 184). Huberman *et al.* (2013) clarify this assertion by suggesting that qualitative research usually involves a small sample of people who become the focus of an in-depth study. While none of the authors venture to provide any form of quantified guidance on the sample size of a qualitative study, Creswell (2013, p. 239) suggests the following guidelines regarding the sample size for qualitative research:

- In narrative research, a sample size of two would be adequate;
- In phenomenological research, a sample size in the range from 3 to 10 is advocated;
- In grounded theory research, a sample size of 10 to 30 is advocated;
- In case study research, there should be a study of at least four to five cases.

While these guidelines apply to general qualitative research, they also have a strong resonance with a qualitative study that adopts a phenomenological approach. The preceding claim is based on the guidelines suggested by Vagle (2016) for the sample size for a phenomenological study. Basically, the researcher has two options. The study could involve spending substantial time with one or two participants over a prolonged period or spending relatively little time with ten to fifteen participants. The choice between these alternatives is an intuitive one and is left to the discretion of the researcher. Whilst these quantification measures merely provide a set of guidelines, Creswell goes on to suggest that the data gathering exercise should continue until no new insights in the main phenomenon of the study is revealed, or a point of data saturation is achieved.

An initial sample of 12 software professionals was used for the purpose of the current study. The criteria used in the selection of the cohort of software professionals is that they should have had at least 5 years of experience in the capacity of a software developer or manager of the software development process in an organisational context and at least 2 years of experience of working in an environment where agile software development methodology (ASDM) is implemented. This criterion-driven phase was supplemented by an opportunistic phase where the researcher was able to use the initial sample as a lead onto other practitioners who met the study's main criterion, a strategy that Huberman *et al.* (2013, p. 31) refer to as "conceptually driven sequential sampling".

The Sample Size and Selection Criteria

As Gill (2014) has suggested, the decision to opt for a phenomenological research approach is based on the imperative to search for the "essences" (P. 5) of the subjective experience of a phenomenon. Malone (2014, p. 42) reinforces the preceding suggestion by commenting that phenomenology "…places a primacy over participant's experience over established theory". Hence the reliance on purposive sampling is crucial because it allows the researcher an opportunity to select participants who can offer a rich insight into the phenomenon of the study.

With regards to the choice of samples size, guidance is obtained from a seminal paper on sample size selection for a PhD study that uses the phenomenological approach by Mason (2010). The essence of this study is to provide knowledge of previous studies that have used phenomenology successfully. In the context of a phenomenological study, success alludes to the potential for the study to obtain data saturation (no new information emanates from the study's respondents) as early as possible. According to Malone, 68% of the phenomenological studies successfully used a sample size that ranged from 5 to 25. With reference to the current the choice of sample of 16 falls well within this range.

With regards to the level of experience of the respondents for the study, guidance is obtained from a study by Malone (2014). In this study, Malone used phenomenology to ascertain the experiences of Scrum Masters when it comes to adaptation of Scrum methodology in an organisational context. In this study, the average number of years of experience by the respondents was 4.7 years. Although Malone prescribed the minimum number of years of experience as 1, much of the information rich data was generated by respondents who had in excess of 2 years of experience. It was also established that the more experienced respondents (between 2 and 5 years) provided greater insight into the adaptation of Scrum methodology. Aligned to this outcome, the current study prescribed a minimum of 5 years of experience of working in the domain of general software development and 2 years of experience in the domain of agile software development. These specifications fall within the parameters reported in the Malone study that provided information rich data and enhanced the prospect of data saturation.

4.3 Method of Data Collection

According to Leedy and Ormrod (2005), the main method of data collection for a phenomenological study is unstructured interviews. The preceding claim is somewhat endorsed in Corbin and Strauss (2014). However, Corbin and Strauss do concede that while the unstructured interview may yield the richest source of data, it could also lack focus and consistency. The main source of concern is that the participants may not be too responsive, in which case the researcher has the arduous task of maintaining the continuity of the interview. A possible strategy to mitigate the afore-mentioned issue is to use a semi-structured interview approach. Although this strategy is somewhat restrictive, the subjects of the study are given an opportunity at the end of the interview to add any other data that they may perceive to be relevant to the study. At this juncture, researchers may also ask additional questions to add some clarity to the discussion. The dualism inherent in the semi-structured interview, where the interviewer has the luxury of resorting to prescribed questions as well as the flexibility to deviate and explore new concepts that may appear during the course of an interview, is what makes the semi-structured interview a viable alternative. Based on the argument presented in the preceding discussion, coupled with the suggestion by Glesne (2015) that qualitative researchers generally make use of semi-structured interviews, the researcher adopted the strategy of using a semi-structured interview.

4.4 The Interview Questions

Corbin and Strauss (2014) are of the opinion that in qualitative research, a researcher may resort to the concepts that formed the essence of the literature review in order to formulate questions for a semi-structured interview. This is important because the questions will indicate the overall intent of the research and convey an image of professionalism to the stakeholders in the researcher's environment. Aligned to this suggestion, the questions that guided the semi-structured interview for the current study were based on a content analysis of the literature review.

Glesne (2015) does however provide valuable insight into the operational phase of the interview process. According to Glesne (2015, p. 96), the questions that are used to guide a semi-structured interview "…are not set within a binding contract" and may be viewed as a tentative set of questions that may be termed as the set of best effort questions. Based on the responses from the initial set of interviewees, the "'best effort' questions may be altered to incorporate glaringly missing concepts or to remove questions that elicited answers that were lacking in depth. Further guidance with regards the interview questions is provided by Corbin and Strauss (2014) who suggest that the questions underpinning a semi-structured interview should consist of technical and non-technical content.

While the preceding guiding philosophy provides a rationale for the researcher to exercise discretion and also imparts a measure of autonomy in the design of the questionnaire, the overall philosophy underpinning the questionnaire design is based on an interview that the researcher conducted with Grady Booch (see Booch (2012)). One of the main themes emanating from this discussion is that software development is a socio-technical activity and any study conducted in this domain needs to incorporate both the social and technical perspectives. This advice aligns quite well with the Corbin and Strauss perspective and it is around this theme that a phenomenological instrument was developed to examine software practitioners' experiences of using agile methodology. The dictates of phenomenological theory (discussed in Section 3.5.3) assumed precedence in terms of the overall design of the questionnaire. However, the inner aspects of the questionnaire contained a software engineering orientation that is based on factors that influence the success/adoption of agile software development in an organisation. There have been many literary contributions in this regard and one of the initial forays into this domain of software engineering research was conducted by Chow and Cao (2008). This study was initiated with a comprehensive literature review to identify pivotal factors that influenced the success of software projects with a specific focus on agile software development methodology in an organisation. The literature review culminated in the development of a conceptual framework that identified the main factors that influenced the success of agile software development methodology. These are the organisational context, the people/stakeholders involved with the system, the technical aspects of the development methodology, the business and project management processes that provide a context for the system and the type, complexity and scalability of the system being developed.

These factors have a strong resonance with the factors identified in studies with a similar agenda and methodology, conducted by Dybå and Dingsøyr (2008) and McLeod and MacDonell (2011). The broad classification of factors listed have also been endorsed in empirical studies conducted by Lalsing *et al.* (2012), Nguyen (2016) and Kropp and Meier (2015). Each of these studies used a conceptual model that had a strong resonance with the Chow and Cao (2008) model (illustrated in Figure 4.1 with a slight adaptation with regards to the terminology used). The

Chow and Cao model is used as a conceptual framework to guide the questionnaire content from an operational perspective.

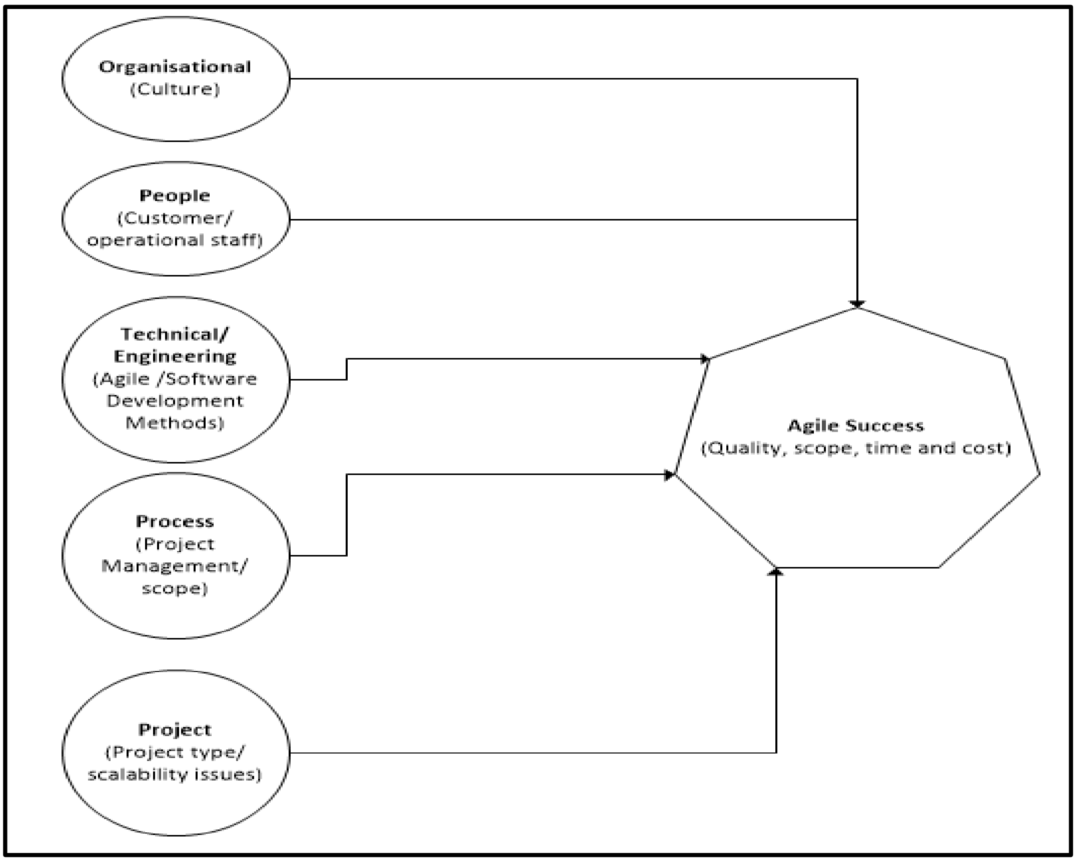


Figure 4.1: An Adaptation of the Agile Success Factors model from Chow and Cao (2008)

The design of the interview questions for the current study were critiqued by two lecturers from the Information Systems & Technology (IS&T) Department at the University of KwaZulu-Natal (UKZN). Each of the lecturers have extensive experience in the use of agile software development methodology by virtue of their involvement in the capstone IS student project. The capstone IS project, which forms a substantial component of the assessment for final year IS&T undergraduate students at UKZN, is implemented using a predominantly Extreme Programming (XP) approach. A discourse on the use of agile methodology to underpin the student project at UKZN is presented in Ranjeeth *et al.* (2013). A significant aspect regarding the use of the agile approach at UKZN is that the overall methodology has a strong Waterfall flavour interspersed with many of the XP methods. Staff at the IS&T department at UKZN are currently deliberating upon a transition to a Scrum based approach. The importance of highlighting the dynamics of the context provided by the academic staff at UKZN is that these staff members have sufficient knowledge of the dominant methods and methodologies that underpin agile software development. From an academic perspective, they have also been involved in the lecturing of the Software Engineering course at postgraduate level. A significant component of the software engineering domain is academic and practitioner-based discourse on the use of current software development methodology to underpin software development in the professional sector. Hence, input from these staff members regarding the design of interview questions were pivotal.

4.4.1 The Need for a Pre-Questionnaire

One of the outcomes of the deliberations regarding the questions that underpinned the interviews for the study was the observation that many of the questions would invariably elicit a structured response that was essentially dichotomous in nature. These questions were aimed at eliciting a response from interviewees with regards to the value that was attached to the core set of agile methods that collectively formed the basis of agile methodology. These questions were classified by the reviewers as potentially routine and would best be administered using a survey-based approach. In order to streamline the interview process, a pre-questionnaire was devised to precede the interview.

The pre-questionnaire contained a series of Likert Scale type questions that served a dual purpose. The primary purpose of the pre-questionnaire was to obtain a structured response with regards to the value attached to each of the methods that frame the prominent agile methodologies such as XP and Scrum. The secondary purpose of the pre-questionnaire was to create a context for the interview session that was planned as a follow-up to the pre-questionnaire component of the engagement with the interviewees.

From a methodological perspective, the survey type pre-questionnaire is classified as quantitative in essence whilst the interview is qualitative. According to Venkatesh *et al.* (2013), the use of diverse methods in IS research can only contribute towards a richer, deeper outcome to the research process. From a purist perspective, the current phase of the study makes use of a multimethod approach but has a predominant qualitative worldview. This approach is further elaborated in Venkatesh *et al.* (2013) as one that is pragmatic, embodying a belief that the "…dictatorship of the research question" (p. 37) takes precedence over the imperative to conform exclusively to an existing methodological paradigm. Aligned to this theory of pragmatism, the primary objective of the pre-questionnaire was to obtain a quick and instinctive response to aspects of software development that have played a prominent role from a software development methodological perspective.

The secondary objective was to use the responses obtained in the pre-questionnaire as a catalyst for further discussion via the interviewing phase of the data collection exercise. The main aspects of software development methodology (discussed in the literature review) that comprised the pre-questionnaire were the following:

- Waterfall methodology;
- Iterative and incremental development;
- The use of a Big Design Up Front (BDUF) strategy;
- The importance of using analysis and design models such as DFDs, ERDs, structure charts, user stories, use case modelling, class and sequence diagrams;
- The use of workflow visualisation tools such as Gant & Pert Charts as well as the Kanban Story Board;
- The importance of XP techniques such as pair programming, test driven development, the availability of an on-site customer, continuous integration and code refactoring;
- The importance of Scrum based techniques such as the product backlog, the concept of a sprint, daily scrum meetings, time boxing,

the maintenance of a sprint backlog, a sprint review meeting, sprint retrospective meetings and a burn-down chart.

The strategy used to elicit opinion on the software development methods, listed above, was to make use of a Likert scale design for the pre-questionnaire. The surveying technique is a commonly used research strategy to obtain knowledge of attitude or opinion towards issues in the domain of software engineering (Pfleeger & Kitchenham, 2001). There are two main types of survey techniques. These are the commonly used unsupervised survey technique or the semi-supervised technique. In the semi-supervised version, the researcher engages with the respondents during their interaction with the survey instrument so that the researcher is able to provide an explanation of the rationale behind the questions asked and to provide a platform for the respondent to provide additional insight pertaining to the survey question.

In order to uphold the qualitative ethos of the data gathering exercise, the researcher opted for a semi-supervised approach for the pre-questionnaire that consisted of Likert Scale type questions. As Rowley (2014) points out, the distinction between questionnaires and interviews is a fuzzy one, because they are both question answering research instruments. Surveys are used when the researcher is trying to establish an overall pattern concerning a phenomenon about which there already exists sufficient knowledge. In the context of the current study, the questions included in the pre-questionnaire focused on routine aspects of software process models as well as agile software development techniques that have been in the domain of software engineering discourse for a substantial period.

The preceding discourse provided some insight into the contents of the questionnaire that will be used to guide the data collection process. However, a 'disclaimer' has to be added with regards to the reliance on the content of the questionnaire. According to Creswell (2012, p. 47), the research process for a qualitative study is "emergent and dynamic". This indictment on the process provides the researcher with substantial flexibility to change questions according to the context of any specific instance of data collection. In accordance with this guiding suggestion, the current study adopted an 'agile' stance to the structure and sequencing of the questions and adapted the questions according to the knowledge that was elicited from the interviewee. There were instances that necessitated the omission, addition and adaptation of the pre-planned questions so that the questionnaire provided a structure that enhanced the prospect of creating an enabling environment where the researcher is able to engage the interviewee in a conversational context. As Creswell suggests, the main idea is to learn about the problem by implementing strategies that enable seamless elicitation of information to enhance the richness of the engagement with the study's respondents.

4.5 The Pilot Study

The pilot study entailed the involvement of four discipline/IT domain experts. The panel of interviewees for the pilot study consisted of 2 academic staff from the Discipline of Information Systems & Technology at University of KwaZulu-Natal (UKZN) as well as two industry professionals who have each had at least 7 years of experience as software developers. One of the two industry professionals has had 10 years of experience as an academic in the IT field as well as six years of experience in the use of agile methodology in the professional sector, spanning two different software development organisations. This combination of academic and professional experience enabled the acquisition of vital insight into the design of the questionnaire so that it had a good balance between academically intrinsic focus areas of software engineering as well as aspects that had a strong resonance with the professional IT sector.

The protocol used in the pilot study was to initiate contact with the members of the pilot study panel via a telephonic conversation where the objectives of the study were explained as well as a query regarding their willingness to participate in the pilot study. In the case of the industry professionals, permission was obtained regarding their willingness to provide data for the actual study. There were separate face-to-face discussions with 3 of the pilot study panel members where a printed version of the interview questionnaire was subjected to their critique. The 4th pilot study panel member was sent an emailed version of the questionnaire and contacted via Skype for a video chat.

From an overview perspective, the pilot study panel were of the opinion that the questions targeted the main issues regarding agile software development methodology. However, a few aspects needed to be noted as omissions, additions or general comments that will arguably ensure that the questions asked during the interview sessions had a good balance and targeted pivotal issues in the domain of agile software development. These aspects are classified according to the designation of the members of the pilot study panel, and are listed below:

From the industry based professionals:

- The use of a pair programming strategy was not done as it is formally implemented in many academic settings (such as the case at UKZN). The comment made in this regard is that pair programming is done more on an ad hoc basis rather as an institutionalised strategy;
- The tracking of project progress is conducted via a plain old whiteboard (POW) strategy rather than making use of any formalised project management tools such as the Gant and Pert Chart, as is the case in an academic context. The POW approach entails the drawing of columns on a whiteboard where project teams document features of an application that 'need to be done', are 'in-progress' and 'have been completed'. This approach has a strong resonance with agile methodology, in particular, the Kanban Storyboard;
- The issue of organisational culture (OC) is not something that many IT professional will identify with from a theoretical perspective. However, they will be knowledgeable about the influence of OC from a pragmatic perspective. A suggestion made in this regard was to provide an explanation of the different 'strands' of OC so that the IT professionals will be able to identify with the theoretical version a lot more easily and provide a more meaningful response in this regard;

- A currently emerging aspect of agile development is the dilemma regarding scalability. Many organisations were resorting to a DevOps-based version of agile methodology. A question/reference to DevOps should be included;
- The original set of interview questions were too long and many industry professionals may not have the time to engage with these issues in a focused manner for such a lengthy period. In addition, there were instances of overlap between aspects that were included as discussion questions. These aspects should be conflated into a single/follow-up question that fitted in seamlessly into the discussion rather than being included as a separate discussion points. These aspects included issues dealing with design/architecture as well as iterative and incremental development.

From the academic representatives:

- *It would be 'nice to know' the actual paradigm of development that was followed in industry*. This comment was a reference to the use of object-oriented (OO) development, a classical/structured approach or a hybrid approach. This comment was deemed to be pertinent to the study because a pure OO approach would entail the use of Unified Modelling Language (UML) which has been claimed to be documentation-intensive and contrary to the objectives of agile methodology (Petre, 2013; Rumpe & Schröder, 2014; Turk *et al.*, 2014). The objective of this knowledge is that it would provide an insight into the modelling/architectural requirements for the implementation an agile approach to software development;
- Issues pertaining to the design of the user interface should be given some sort of coverage in the interview questionnaire. This suggestion has been endorsed by the researcher as a valid one and

it is an issue that has been the subject of extensive deliberations as alluded to in Brhel *et al.* (2015).

While the input obtained from the pilot study was quite useful in ensuring a measure of validity with regards to the content of the questionnaire, the overall design of the questionnaire was guided by the dictates of Rubin and Rubin (2012, p. 6) who suggest that the structure of an interview should revolve around three types of linked questions. These are the main questions, probes and follow-up questions. The main questions attempt to ensure that there is adequate linkage with the research questions, probes are used to encourage the interviewee to continue talking and follow-up questions are used to explore main themes discussed enabling the researcher to elicit more depth from the interview. The questionnaire (see Appendix A) used as a guide for the interview sessions in the current study is structured along the lines of main questions, follow-up questions and probe type questions

The questionnaire content is guided by the suggestion in Rubin and Rubin (2012) that the main questions should emanate primarily from the researcher's knowledge and experience in the study's domain. A secondary source could be the academic literature. However, they do warn that the formulation of main interview questions from the academic literature is not ideal, because "…it will blind the researcher to what is actually out there" (Rubin & Rubin, 2012, p. 134). The academic literature should be used sparingly and questions based on the literature should be worded carefully and simply so as not to convey the idea that the interview entails an examination of the interviewee's theoretical knowledge. Also, in order to obtain an insight into the lived experience(s) of the interviewee, the researcher needs to pose questions that are open-ended and to make use of a technique called *responsive interviewing* (Rubin & Rubin, 2012, p. 4) where the researcher is able to dynamically "…change questions in response to what he or she is learning." This responsive style of questioning has been adhered to in the current study by virtue of the probes and follow-up question that have been included in order to arguably ensure that there is a measure of responsiveness with regards to the questions asked. This also enables the interview process to resemble a *conversational partnership* that the researcher establishes with the interviewee (whom Rubin and Rubin (2005, p. 6) refer to as a *conversational partner*). The probes and follow-up questions that were dynamically generated and converged to a finite set is highlighted in the questionnaire (see Appendix A).

4.6 The Main Interview

Prior to any formal form of engagement, as is required by institutional norm, informed consent has to be obtained so as to ensure that the participants are not forced to participate, but do so on a voluntary basis and are also given a guarantee of anonymity if required. As is suggested in Rubin and Rubin (2012), the committee that oversees the ethical issues that underpin a study needs to be informed that the qualitative nature of the study does not enable the researcher to provide the exact wording or content of the questions that will be asked. However, a set of sample questions may be made available to the committee so as to minimise the prospect of ethical violations with regards to the type of questions that are asked. In order to abide by this suggestion, the researcher made available the main interview questions and follow-up questions (see Appendix A) to Committee for Research Ethics at UKZN. The ethical clearance certificate for the qualitative phase of the study can be referenced in Appendix B.

4.6.1 The Study's Participants

As Creswell (2012, p. 149) points out, the important criterion in identifying respondents in a phenomenological study is to ensure that they are "…individuals who have all experienced the phenomenon being explored and can articulate their lived experiences." For the purpose of the current study, respondents have been identified as software professionals who have had at least 5 years of experience in the domain of software development/management with the proviso that they have had at least 2 years of experience in the domain of agile software development methodology. Hence the use of purposive sampling becomes the most appropriate choice of sampling technique. The main foray into the data collection exercise was based on the researcher's knowledge of the extensive and highly informed use of agile methodology in one of the 4 of the major banking institutions in South Africa. Coupled with this initial contact and an internet based search for software development organisations that subscribed to the use of agile software development methodology, a total of 36 prospective participants were identified for the study. Each of the participants was contacted telephonically to explain the objectives of the study and to determine their willingness to participate. The objectives of the study were also explained via an email. The consent form as well as the pre-questionnaire was sent as an attachment. Eighteen of the prospective respondents replied via return email with responses to the questions asked in the pre-questionnaire. From this group, only 12 of the respondents indicated their willingness to participate in the interview session. The interview sessions were scheduled to take place in the time period from October 2016 to January 2017. Due to the unavailability of many of the respondents during the planned period, the bulk of the interviews were conducted from December 2016 to April 2017. During this 'core data collection' period, many of the interviewees suggested the names of colleagues who would add value to the knowledge on agile software development, based on their experience and expertise in that domain. This iterative approach to sampling is usually recommended in a qualitative study to optimise the prospect of obtaining theoretical saturation, thereby enabling the researcher to make valid conclusions (Huberman *et al.*, 2013). A final set of 16 interviews were conducted. The interview schedule is presented in Appendix C. The interview schedule includes the profile of the interviewees with regards to the type of organisation that they currently worked in, the capacity in which they have served in the organisation as well as the total number of years of experience in the domain of software development and the number of years of involvement with projects that implemented an agile software development methodology.

A significant observation that was made during the researcher's engagement with the subjects of the study is the intensive effort that the banking sector in South Africa has made to invoke strategies that enable an improvement in the software development process. A major focus of this effort has been in the domain of agile software development methodology. The researcher was provided with an opportunity to meet with many of the members of the software development teams in the banking sector who were being 'groomed' by agile coaches on various aspects of agile methodology. The 'richness' of the dialog with software practitioners in the banking sector is the main reason for the prominence of representatives from this sector in the sample selected for the analysis (illustrated in Figure 4.2).

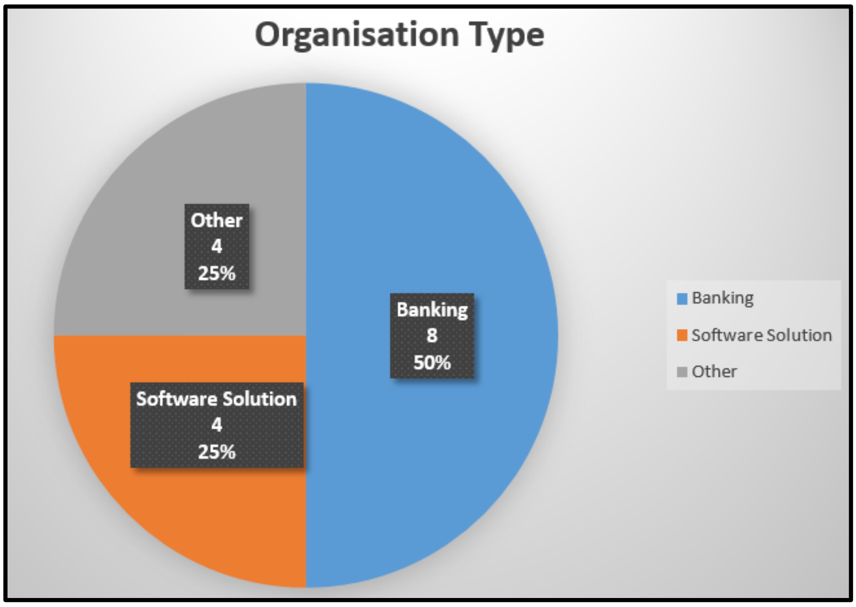


Figure 4.2: Type of Organisation Represented by the Interviewees

As can be observed in Figure 4.2, the main business domain represented by the interviewees is the banking sector. A total of 8 interviews were done with software practitioners from all 4 of the largest banks in South Africa, constituting 50% of the total number of interviews that were conducted. Four interviews were held with software practitioners who belong to organisations that provide a bespoke software solution service to various organisations in South Africa. The remaining 4 interviews (labelled as 'other' in Figure 4.2) were conducted with practitioners from the agricultural sector, the motor industry, the petro-chemical industry and a national logistics organisation. In each of these organisations, agile methodology has been used extensively for software development thereby enabling an informed response by the software practitioners of the value of agile methodology from a phenomenological perspective.

An important aspect of a phenomenological study is that the experience of the respondents in the domain of inquiry is crucial to ensure that the interaction between the researcher and the respondent yields meaningful data. As discussed previously, the 2 compulsory minimum requirements were that the respondents must have at least 5 years of experience in the general domain of software development, in the capacity of a software developer or manager, as well as 2 years of experience of working in an agile software development environment. The respondents for the current study were well within these established parameters as is verified in the subsequent narrative.

From the total of 16 interviews that were analysed, the number of years of experience of the interviewees in the professional software sector has a range from 5 to 30 years. One of the respondents had 30 years of software development and management experience. However, the data value of 30 was recognised as an outlier (illustrated in the Box and Whisker plot of Figure 4.3) and removed from the computation of the measures of central tendency for the number of years of experience that the respondents have in general software development.

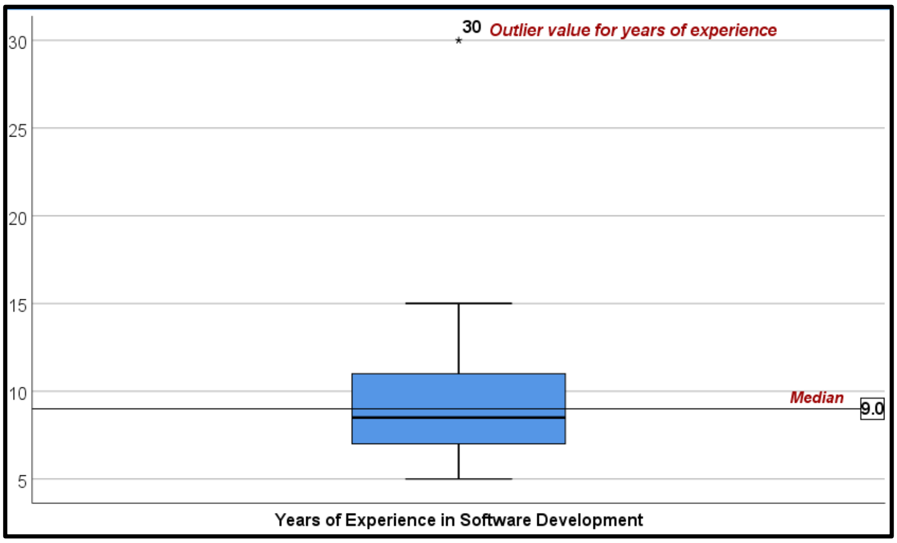


Figure 4.3: Box and Whisker Plot showing Outlier for Years of Experience

The *average* number of years of experience of the interviewees in general software development is 8.8. As illustrated in Figure 4.4, the number of years of experience in general software development for the bulk of the interviewees is in the range from 7 to 10 years.

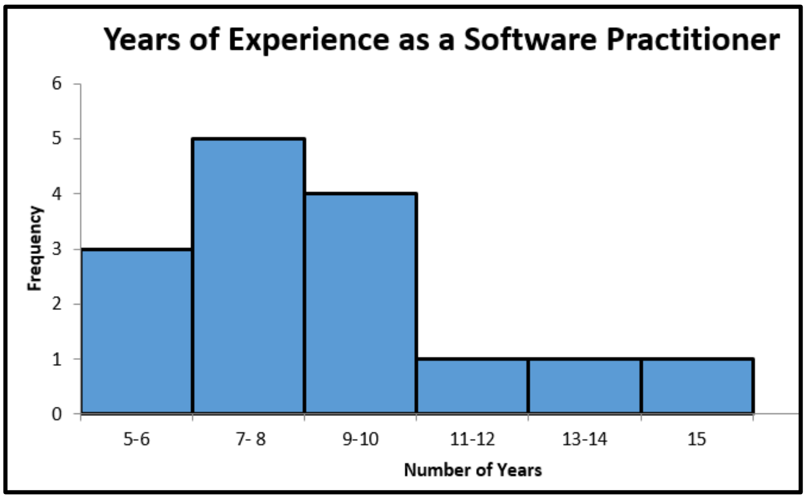


Figure 4.4: Number of Years of Experience as a Software Practitioner

In terms of the number of years of experience in the use of *agile* software development methodology (ASDM), the range is from 3 to 9 years with an average of 5.9 years and a median of 6 years. The mode with regards to experience in the use of ASDM is 5 years and as illustrated in Figure 4.5, the number of years of experience for the bulk of the interviewees is in the range from 5 to 8 years.

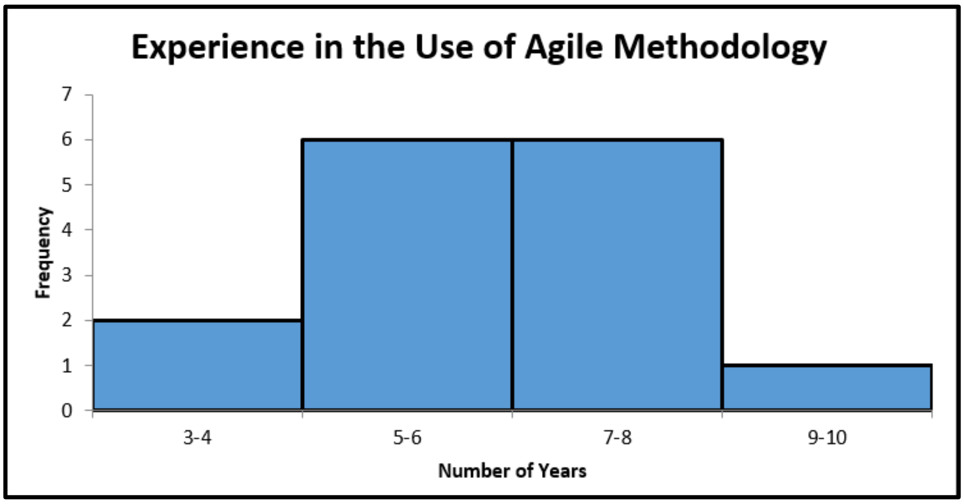


Figure 4.5: Number of Years of Experience in Using Agile Methodology

It should be noted that a large number of respondents (as indicated in the full interview schedule in Appendix C) started off their careers as software developers and later progressed to roles as business managers, systems analysts and in some instances, solution architects. As part of their experience, they have all been involved in the development of software systems that made use of the traditional waterfall methodology, agile methodology and a hybrid of agile methodology and waterfall methodology. Hence the sample used in the current study had all the attributes to provide a multi-dimensional perspective on their experiences in software development and agile methodology in particular.

It should also be noted that the interview schedule in Appendix C makes reference to 20 interviews that were recorded as part of the study's data corpus. As mentioned previously, 16 of the interviews were transcribed and analysed as part of the study's qualitative analysis phase. The role played by the remaining 4 interviews is explained below.

- The first interview conducted for the study was done with IBM Research Fellow, Grady Booch. The objective of this interview was to obtain a focus area for the study from a highly respected expert in the domain of software engineering;
- Three interviews were conducted with software engineers who have expertise in the domain of agile methodology and the operations phase of the development lifecycle (the DevOps dimension). These interviewees were well-placed individuals who were invited by the researcher to verify the strengths, weaknesses and ideas on how agile methodology could be integrated with the operations phase. These 3 interviews were conducted after the main set of exploratory interviews were conducted and were included as part of the synthesis phase of the qualitative data analysis.

4.6.2 The Interview Protocol

The protocol adopted for the study is structured along the sequence of activities for qualitative research as suggested in Creswell (2013, p. 192). The sequence entails identification of a set of open ended questions, identification of the panel of interviewees who have experience in the phenomenon being studied, engagement with members of the panel either individually or as a focus group discussion, audiotape and transcribe the interview. For the purpose of the current study, initial contact with all potential respondents was made via an email request followed by a telephonic conversation regarding the logistical arrangements for an interview. The initial email contained the research consent form as well as the pre-questionnaire that was used to set a context for the dialogue on agile methodology. The respondents were asked to fill in their demographic details including their job profile and experience in the IT domain. In most instances, these consent forms and pre-questionnaire were emailed back to the researcher together with a note indicating the respondent's willingness to participate in the interview. In a few instances, the consent form was physically collected by the researcher during the interview session.

Nine interviews (from the main data corpus) were conducted as face-to-face interview sessions that were conducted at the interviewees' workplace. The remaining 7 interviews were conducted via Skype video. All interview sessions were recorded as audio sound files by a professional external recording device. A backup recording was also made by the built-in Skype recorder as well as the recording tool that is found as a utility of the Windows desktop operating system. As suggested in Creswell, the researcher recorded the date, time and location of each interview session and also made notes of points of emphasis and visual expressions of the interviewee during the interview session.

The interview itself entailed a few 'ice-breaker' questions and comments before the main opening question that focused on the interviewee's experience in the use of agile methodology with reference to specific instances where it worked well and instances where it proved to be problematic. The initial engagement was used as a platform from which additional probing questions were asked in order to elicit meaningful insight into the interviewee's experience of agile methodology and to maintain continuity with the questions so that a conversational demeanour was maintained. The interview was concluded by thanking the interviewees for their time coupled with an enquiry with regards to their availability to provide clarity on issues that were discussed in the interview. A query was also made to establish contact details of other potential respondents whose input would be of value to the study.

At the conclusion of each interview session, the recordings were transcribed as soon as possible so that any part of the recording that may have been unclear would be easily recalled by the researcher based on the actual interview. The transcriptions varied in the number of pages per interview. The average length of the interview sessions is 52 minutes. The total number of transcription pages is 235 with an average of 14.6 pages of text per transcription.

5.0 QUALITATIVE DATA ANALYSIS AND PRESENTATION

5.1 Introduction

Qualitative data analysis consists of preparing and organising the data and then reducing the data into themes through a process of coding and code condensation, Creswell (2013, p. 179). A summarised form of the analysed data is presented in the form of figures, tables, or a discussion that provides a rich textural description of the salient issues that are conveyed by the data corpus. The main purpose of qualitative data analysis is to develop concepts that enable an understanding of a phenomenon by leveraging off the experiences, opinions and meanings attached to that phenomenon by members of a social system. The phenomenon under inquiry is usually not easy to understand via quantitative research methodology. It is best analysed by invoking methods that can interrogate rich textual descriptions of the phenomenon by subjects who can relate and explain their experience of the phenomenon. Analysis entails an iterative, inductive engagement with the 'raw data' in order to extract themes from the textual content. These themes are coded and classified so that so that prominent categories can be identified to enable conceptualisation of the research problem (Pope & Mays, 1995).

The preceding narrative on quantitative data analysis resonates with the research design and analysis that is used in the current study. The first phase of the analysis involves content analysis to reduce the raw interview data into categories that will enable conceptualisation of the practitioner's experience and opinion of agile software development methodology. However, cognisance has to be accorded to the cautionary advice from Myers (1997) that qualitative data analysis is not as structured and prescriptive as is the case with a quantitative study. One aspect of this complexity is attributed to the relationship between the data gathering and the data analysis phases. In quantitative research, there is a clear distinction between both phases of the research process. However, in a qualitative study this distinction is somewhat blurred, especially in the context of a hermeneutic study where the researcher's assumptions and biases may influence the data gathering phase by virtue of the questions that are posed to the subjects of the study. In such a situation the data and the analysis has a bidirectional relationship where the data informs the analysis and the analysis is used to arguably ensure that the correct data is gathered. Much of the complexity in a qualitative study emanates from the imperative to adopt varying degrees of intensity with regards to data collection, analysis and interpretation at the different stages of the study. According to Myers, textual analysis of qualitative data resembles a 'hermeneutic circle', a reference to the dialectic between comprehension of the text from a holistic perspective as well as the interpretation of the individual parts of the text. In the context of an information systems related study, Myers (1997) suggests that the holistic interpretation needs to incorporate the relationship between the people, the organisation, and the technology. A complementary technique is semiotics, which entails an analysis of the words in a transcript so that individual words may be assigned to categories which are then interpreted in terms of the frequency with which these categories present themselves in the text. The interpretive aspect of a semiotic approach is referred to as content analysis, a qualitative research technique where "…the researcher searches for structures and patterned regularities in the text and makes inferences on the basis of these regularities," (Myers, 1997, p. 12).

The opening narrative is used to introduce some of the terminology and techniques that are synonymous with qualitative data analysis. However, the operational aspects of conducting qualitative data analysis for a study that is aligned to phenomenology is guided to a large extent by the writings of Creswell (2012), Moustakas (1994) and Saldana (2009). Guidance in this regard is also obtained from the works of grounded theory specialists such as Corbin and Strauss (2014) and Huberman *et al.* (2013). The rationale behind this approach is that there is a lot of commonality between the various types of qualitative data analysis and it is an acceptable practice to leverage off an array of techniques that serve the ultimate objective of attaching meaning to qualitative data.

5.2 Framework for Phenomenological Qualitative Data Analysis

The qualitative data collection phase entailed the implementation of an interviewing strategy that invoked a bracketing and a hermeneutic approach. The bracketing phase of the interview consisted of the opening questions where the interviewees were asked to talk openly about their experience in the use of software development methodology in general and a follow-up question that made specific reference to agile software development. The subsequent questions in the interview were aligned to the hermeneutic phenomenological approach where the researcher's knowledge and experience of software development issues were used to enhance the prospect for a coherent conversation-style to be adopted for the interview so that the richness of the engagement is enhanced in a natural way. This strategy is aligned to the guidance on phenomenological interviews advocated in Creswell (2013) and Moustakas (1994). The responses to the first two questions are pivotal in guiding the analysis phase that seeks to establish an understanding of the common experiences, by the interviewees, of the phenomenon under inquiry. The researcher is required to go through the data and identify statements and extracts of anecdotal evidence that are used to develop 'clusters of meaning' from these significant observations so that themes can be identified to provide an understanding of how the interviewees experienced the phenomenon. Moustakas (1994, p. 120) refers to this process as "horizontalisation". A conceptual framework for the analysis of the qualitative data for the current study is illustrated using flowcharting notation in Figure 5.1. The framework is an adaptation of the Van Kaam method for phenomenological data analysis that is presented in Moustakas (1994, p. 121).

The nomenclature used to describe qualitative data analysis is vast and consists of terminology that is often equivalent in meaning. The terminology used by Moustakas is synonymous with most of the qualitative analysis terminology that is used in Creswell (2013), Huberman *et al.* (2013) and Saldana (2009). This commonality is revealed through an examination of the specific phases illustrated in Figure 5.1.

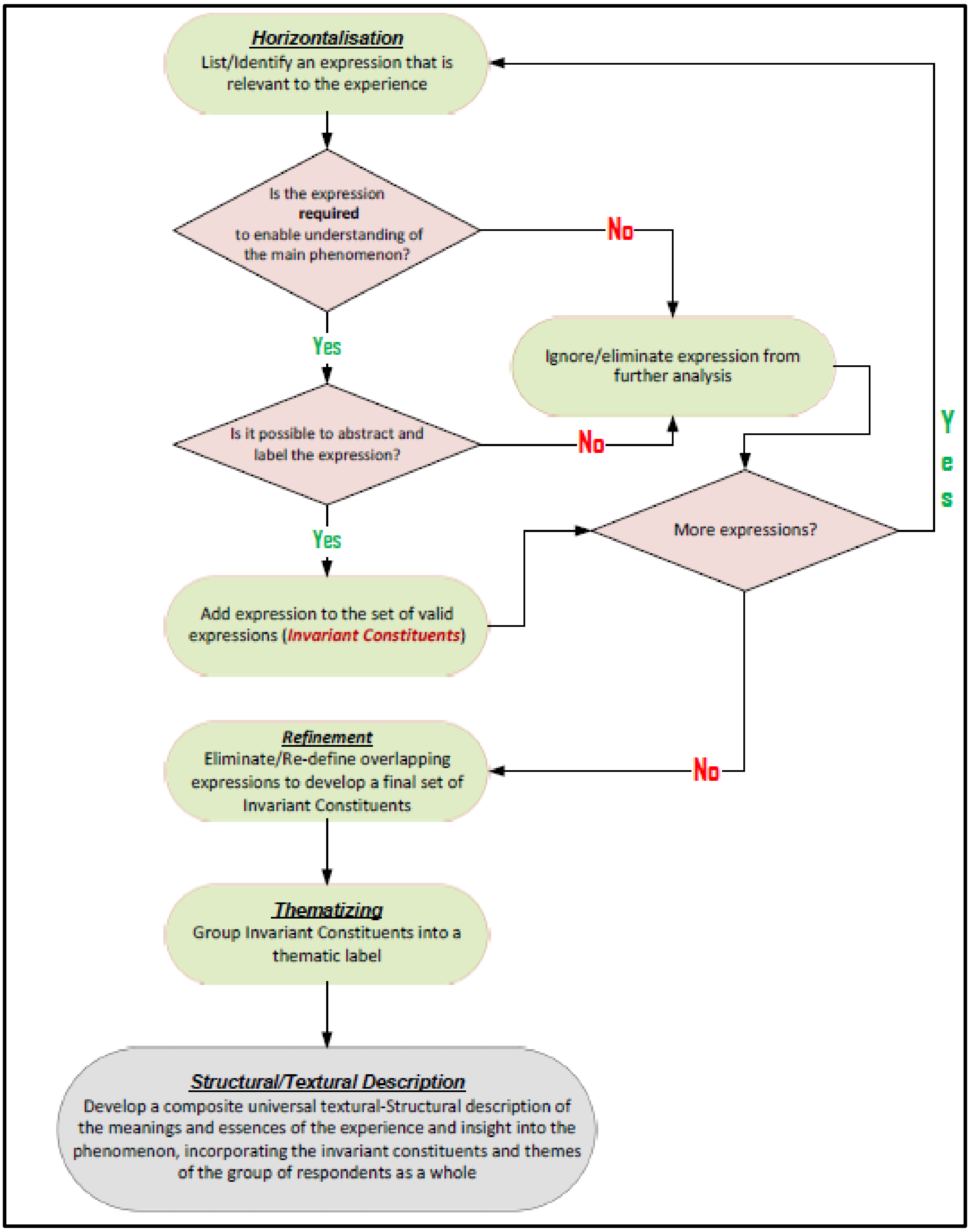

Horizontalisation
List/Identify an expression that is relevant to the experience
Is the expression required to enable understanding of the main phenomenon?
No
Yes
Ignore/eliminate expression from further analysis
Is it possible to abstract and label the expression?
No
Yes
More expressions?
Yes
Add expression to the set of valid expressions (Invariant Constituents)
Refinement
Eliminate/Re-define overlapping expressions to develop a final set of Invariant Constituents
No
Thematizing
Group Invariant Constituents into a thematic label
Structural/Textural Description
Develop a composite universal textural-Structural description of the meanings and essences of the experience and insight into the phenomenon, incorporating the invariant constituents and themes of the group of respondents as a whole

Figure 5.1: Framework to guide the analysis of Qualitative Data

The horizontalisation phase is the first phase of analysis where the researcher engages with the raw data with the intention of creating expressions or codes that serve as an abstraction of the raw data. The objective of this exercise, which Corbin and Strauss (2014, p. 105) refer to as "initial coding" and Saldana (2009, p. 8) refers to as First Cycle Coding, is to reduce and condense the level of detail to enable further analysis. The next phase of First Cycle coding is to create code categories (Corbin & Strauss, 2014, p. 220) where codes are grouped according to the phenomenon that they reference or "chunked" into broad topic areas (Bazeley & Jackson, 2013, p. 105).

From a more practical perspective, the activity of coding when using qualitative data analysis software (QDAS) such as Nvivo, entails the creation of nodes where chunks of data that have a similar meaning are coded as a node. Groups of nodes may be categorised under a common parent node thereby creating a hierarchy of node categories.

The next phase which represents a higher level of abstraction is referred to as *thematising*. Themes are uncovered on the basis of an analysis of the codes and categories in order to identify patterns and relationships that may contribute to the answering of the research question (Saldana, 2009, p. 5). Thematising is part of the Second Cycle coding phase that Saldana alludes to. The idea is to reduce or conflate the First Cycle codes into a broader category that paves the way for theory development or a holistic textural description of the data.

In the context of the current study, the preceding coding and thematising methodology (illustrated in Figure 5.1) is followed in order to obtain insight into the phenomenon of agile software development methodology as experienced by software practitioners in South Africa.

5.3 The Coding Phase

A core activity in qualitative data analysis is the coding phase. Huberman *et al.* (2013, p. 72) emphasise the significance of coding by suggesting that qualitative analysis has commenced once a researcher engages in the activity of coding. There are various explanations offered to elucidate the purpose of coding (e.g. Huberman *et al.*, 2013; Saldana, 2009) From these explanations, it becomes apparent that coding is part of the data reduction or data condensation process to enable the researcher to succinctly capture the essence of the volumes of data that is typically gathered in a qualitative study. According to Huberman *et al.* (2013, p. 71) codes are labels that are used to categorise data as well as to convey the essence of the data by making use of meaningful, descriptive names. The codes are also a convenient strategy to enable the researcher to quickly retrieve and reference 'chunks of textual transcripts' to set the stage for further analysis and the development of a construct or a theory. The process of coding is not an exact science and is largely heuristic, based on the researcher's intuition and careful reading and reflection in order to obtain intimate understanding of the message that is being conveyed in the textual transcripts. Saldana makes reference to 25 approaches that may be followed for First Cycle coding. From this list, the most relevant choices for the current study are the following:

- *Descriptive*: The use of a noun or an expression to succinctly capture the essence of a passage of text. The set expression eventually provides an inventory of topics that serve as an abstraction of the raw data;
- *In Vivo coding*: A popular coding strategy that has the same objective as descriptive coding, but is technically different in the sense that it makes use of short phrases taken from the participant's own language as an initial code;
- *Process Coding*: Entails the use of gerunds (a verb form that serves as a noun) to represent action or interaction sequences in the text.

The First Cycle coding approach is usually quite time consuming, but it adds some structure to the qualitative data. Corbin and Strauss (2014, p. 25) offer some respite by cautioning against an obsession with too much structure and the need to maintain an element of flexibility and fluidity so that intuition and insight from the researcher is not totally ignored.

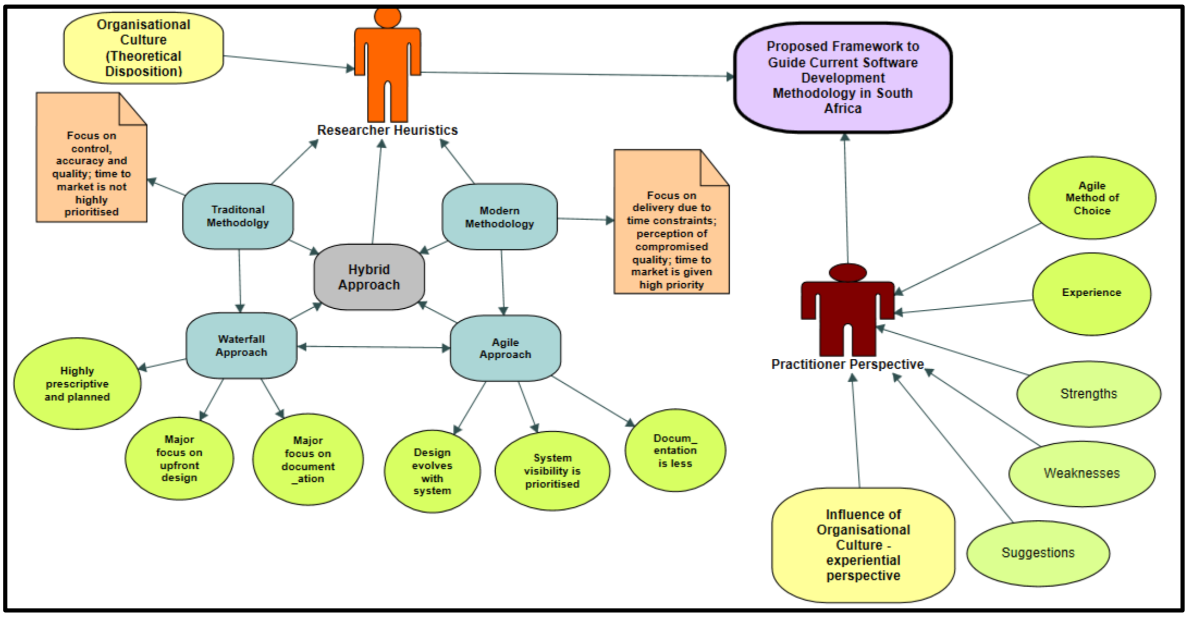
5.3.1 The Use of Qualitative Data Analysis Software

Qualitative data analysis software (QDAS) is used as a supplementary qualitative data analysis tool. It is not meant to replace the 'time-honoured' tradition of manually examining data to establish relationships and patterns.

However, software tools such as the Nvivo software package that was developed to support the qualitative researcher, serves as an ideal mechanism to "manage" the data thereby enabling the researcher to focus on the meaning conveyed by the data (Bazeley & Jackson, 2013, p. 2). The current study adopts a strategy of using the Nvivo 11 Professional Version for the qualitative data analysis based on the premise that Nvivo will provide an enhanced capacity for recording, sorting, matching and linking of qualitative data while also maintaining access to the source data or contexts from which the data have come.

The Initial Mind Map

At the outset, Bazeley and Jackson (2013, p. 28) advise that the qualitative researcher should develop an initial concept/mind map that documents assumptions and also clarifies the conceptual framework that underpins the study. From a QDAS perspective, this is quite beneficial because it allows the software to make comparisons between emerging concepts and the initial pre-conceptual constructs that are introduced by the researcher at the start of the analysis process. For the current study, preference is given to a graphical version of such a pre-conceptual map. This is illustrated in Figure 5.2



Organisational Culture (Theoretical Disposition)
Researcher Heuristics
Proposed Framework to Guide Current Software Development Methodology in South Africa
Focus on control, accuracy and quality; time to market is not highly prioritised
Traditonal Methodolgy
Modern Methodology
Focus on delivery due to time constraints; perception of compromised quality; time to market is given high priority
Hybrid Approach
Waterfall Approach
Agile Approach
Highly prescriptive and planned
Major focus on upfront design
Major focus on document _ation
Design evolves with system
System visibility is prioritised
Docum_ entation is less
Practitioner Perspective
Agile Method of Choice
Experience
Strengths
Weaknesses
Suggestions
Influence of Organisational Culture - experiential perspective

Figure 5.2: A Pre-Conceptual Mind Map

The main constructs of the pre-conceptual mind map illustrated in Figure 5.2 are the pre-conceived heuristics (largely emanating from the literature review) of software development that are used by the researcher to add structure to the engagement with the software practitioner. The researcher and practitioner perspectives are guided by socio-technical elements represented by organisational culture and software development methodological rigour that has a strong technical orientation. The essence of the mind map is that it has to make reference to the traditional approach to software development epitomised by the Waterfall approach so that a comparison standard can be created. There also has to be a reference to agile software development methodology because it epitomises current software development practice and it is a core aspect of the current study. Based on the outcome of the literature review, the hybridisation of agile methodology plays a prominent role in the actual implementation of the methodology. The traditional, modern and hybrid approaches to software development provide the terms of reference that may be used to optimise the insight obtained from the engagement with the software practitioner, paving the way for a synthesis phase that produces a model/framework that enhances existing software process models.

5.3.2 Initial Coding

Initial coding is a technique that is a subset of Saldana’s First Cycle (see Saldana, 2009) coding methodology where the researcher engages in a process of breaking down the ‘mass’ of qualitative data into manageable parts. This is referred to as the process of conceptualising the raw data into a higher level of abstraction which represents a meaningful form of data reduction. Creswell (2013) warns of the challenges associated with qualitative data analysis because of the volume of data that needs to be analysed. In order to manage this process, Creswell eloquently suggests that qualitative data analysis conforms to a general contour that is referred to as a “data analysis spiral” (Creswell, 2013, p. 182). The spiral analogy is used to convey a methodology where the researcher moves iteratively between the phases of data collection, data capture, data analysis and reporting. This approach is used to decipher the complexity that is usually found within the confines of the voluminous textual data. Aligned to this spiral approach, the data analysis for the current study is conducted via a strategy where the first 9 interview transcripts are analysed as an initial foray into the data analysis phase. The objective of this exercise is to enable the researcher to obtain an initial sense of the main concepts that emerge from the initial set of interviews. This knowledge will guide the researcher to achieve the purposeful sampling objective as well as to focus on interview questions that provide a better insight into the prominent concepts that emanate from the initial foray into data analysis. This strategy of selective sampling and interviewing will enhance the prospect of achieving data saturation in an intelligent way.

The transcripts from each interview were entered into Nvivo in chronological order. Each transcript was linked to a memo that incorporated field notes made by the researcher during the course of each interview. These field notes served a dual purpose. They were used to make reference to the interviewee's organisational context in terms of the domain of the organisation as well as the capacity that the interviewee served in the organisation. A reflective entry was also made in the memo providing details regarding the salient ideas that emerged from the interview. This memo provided valuable guidance during the transcription process because elements of the interview that were not well recorded or aspects that were not coherently expressed by the interviewee were supplemented with comments made in the memo. In cases where both instruments, the interview recording and the memo did not achieve the objective of providing the insight required, an email was sent or a telephone call was made to the interviewee just to clarify the 'grey area' of understanding. This strategy enhanced the prospect of maintaining a very good level of interactivity with the set of interviewees subsequent to the actual interview itself.

The data analysis process was done both deductively and inductively. The initial mind map illustrated in Figure 5.2 contributed to the deductive aspect of the data analysis by providing an initial set of concepts that are used as categories for the coding phase. The Nvivo nomenclature for data analysis includes the term 'node' to represent these categories. In order to identify a viable set of nodes for the current study, a word frequency count was computed from the analysis of the 1st 9 interview transcriptions. The words that were identified to be the most frequently used was analysed for further meaning by examining the context in which these words were used. These words were augmented with an additional word so that it was an accurate representation of the context in which these words appeared in the transcripts. The process of word augmentation was conducted by running a text search query and establishing the most common words that were found near a specific word. As an example, a text query of the words that were in proximity of the word “developmental” yielded the tree structure illustrated in Figure 5.3.

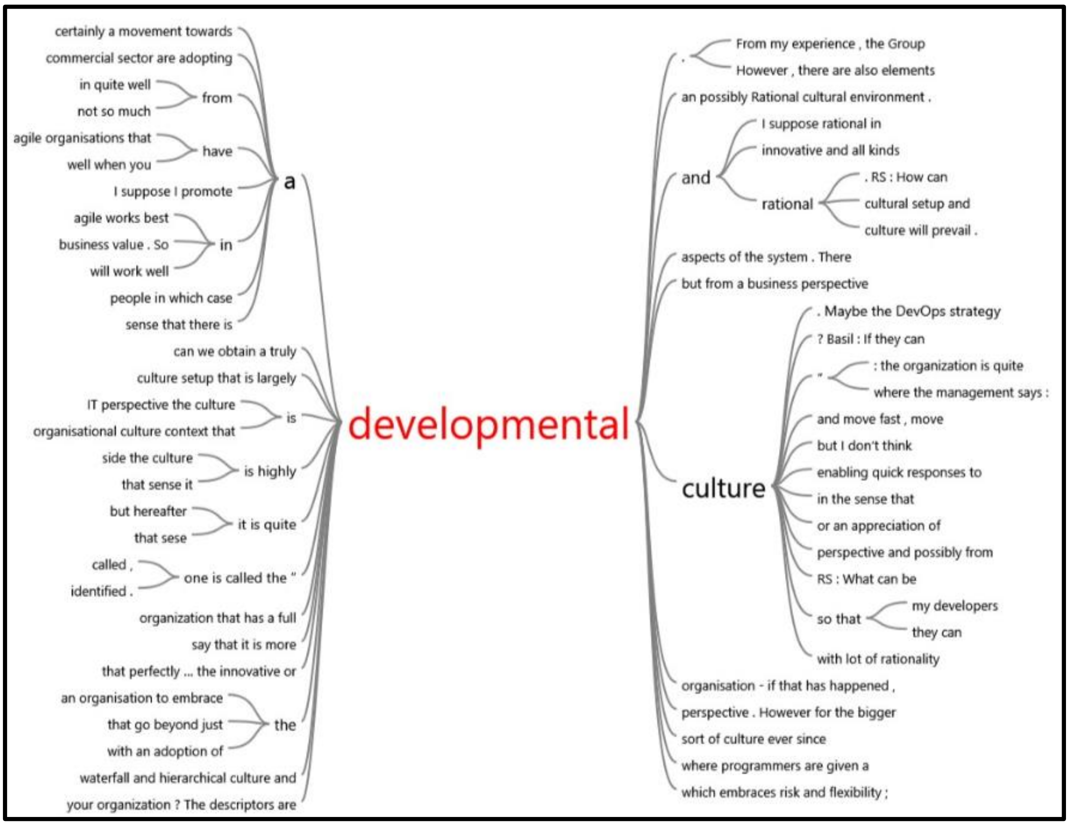
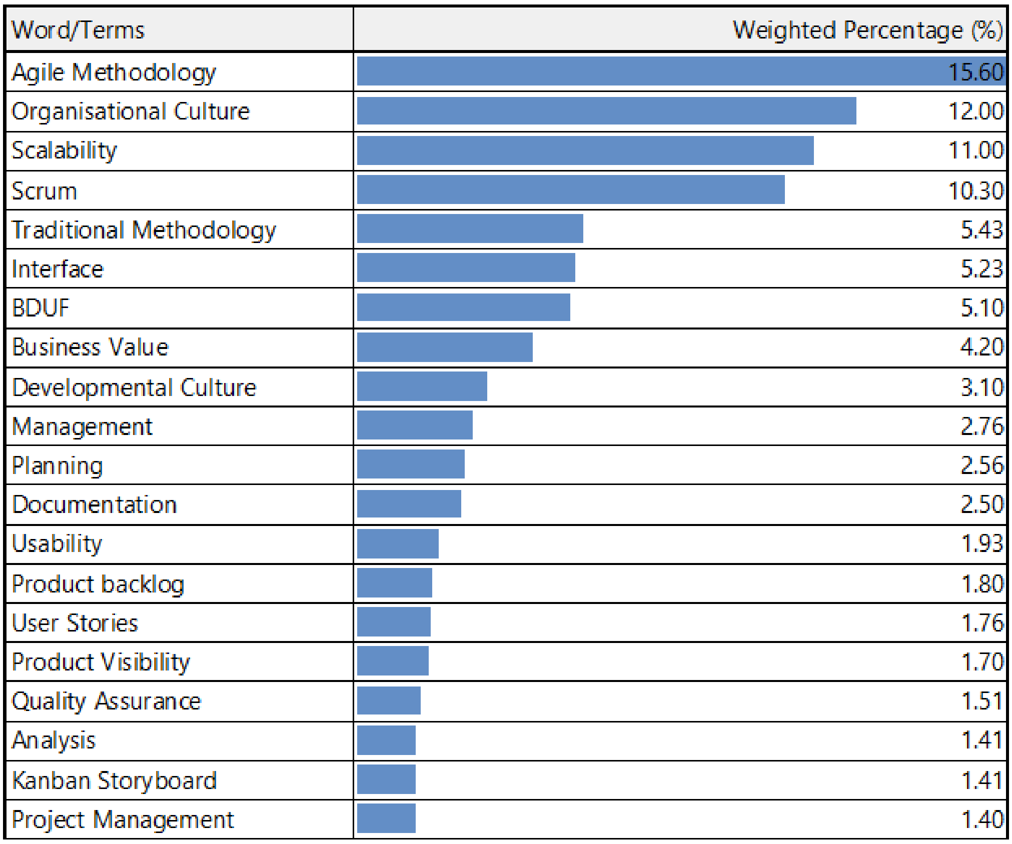

certainly a movement towards
commercial sector are adopting
in quite well
not so much
from
agile organisations that
well when you
have
I suppose I promote
a
agile works best
business value . So
will work well
in
people in which case
sense that there is
can we obtain a truly
culture setup that is largely
IT perspective the culture
organisational culture context that
is
side the culture
that sense it
is highly
but hereafter
that sese
it is quite
called ,
identified .
one is called the “
organization that has a full
say that it is more
that perfectly ... the innovative or
an organisation to embrace
that go beyond just
with an adoption of
the
waterfall and hierarchical culture and
your organization ? The descriptors are
developmental
.
From my experience , the Group
However , there are also elements
an possibly Rational cultural environment .
and
I suppose rational in
innovative and all kinds
rational
. RS : How can
cultural setup and
culture will prevail .
aspects of the system . There
but from a business perspective
culture
. Maybe the DevOps strategy
? Basil : If they can
"
: the organization is quite
where the management says :
and move fast , move
but I don't think
enabling quick responses to
in the sense that
or an appreciation of
perspective and possibly from
RS : What can be
so that
my developers
they can
with lot of rationality
organisation - if that has happened ,
perspective . However for the bigger
sort of culture ever since
where programmers are given a
which embraces risk and flexibility ;

Figure 5.3: Text Query Search on the word ‘developmental’

From the results of the text query search shown in Figure 5.3, it can be established that the context for the word ‘developmental’ is in reference to the culture that prevails in the organisation. Hence the words ‘developmental’ and ‘culture’ were combined in the output of the word frequency count exercise as illustrated in Figure 5.4 below. Also, the words that were deemed to be superfluous to the word analysis exercise were removed from the word frequency calculation. A weighted average of the remaining words was computed and a frequency distribution of the 1st 20 words/terms arranged in descending order of the relative percentage of references that were coded according to this word/term is illustrated in Figure 5.4.



Word/Terms	Weighted Percentage (%)
Agile Methodology	15.60
Organisational Culture	12.00
Scalability	11.00
Scrum	10.30
Traditional Methodology	5.43
Interface	5.23
BDUF	5.10
Business Value	4.20
Developmental Culture	3.10
Management	2.76
Planning	2.56
Documentation	2.50
Usability	1.93
Product backlog	1.80
User Stories	1.76
Product Visibility	1.70
Quality Assurance	1.51
Analysis	1.41
Kanban Storyboard	1.41
Project Management	1.40

Figure 5.4: Preliminary Node Identification Based on Word/Term Frequency Count

The objective of this exercise was to enable the identification of a viable set of nodes that will guide the coding process. This strategy also ensured that the coding process incorporates an inductive approach as well since the coding categories/nodes which were initially pre-defined have been supplemented with the additional nodes identified from Figure 5.4. The additional codes emanated from the raw data, a strategy aligned to the dictates of qualitative data analysis as suggested in Corbin and Strauss (2014). According to Corbin and Strauss (2014,

p. 221), as raw data is captured for analysis, the underlying concept represented by the data needs to be identified as the code or category that represents a higher level of abstraction of the underlying data. Attached to this concept should be a memo that documents the relevance of the concept in the context of the research questions. From an Nvivo data capture perspective, these concepts are represented as nodes.

Preliminary Data Analysis and Presentation

The need to conduct a preliminary data analysis is aligned to the suggestion by Corbin and Strauss (2014, p. 221), that the preliminary analysis is pivotal in establishing a "springboard" for subsequent analysis. This preliminary analysis of raw data coupled with the researcher's observations and recognition of patterns or trends in the data, is a useful strategy to enable convergence of understanding of the main issues relating to the phenomenon of the study.

Based on the analysis of the first 9 interview transcripts, a total of 68 nodes/categories (including sub-categories) were identified. The preliminary use of frequency counts is advocated by Huberman *et al.* (2013) who are of the opinion that such knowledge will enable the researcher to converge the categories identified to a manageable set. For the purpose of the current study, a hierarchical chart is used to obtain a broad perspective of the most influential categories based on the number of references that were attached to these categories. An adjusted (for illustration purposes) version of this chart showing the main nodes/categories that received the most number of references is illustrated in Figure 5.5. It should be noted that the display area of the rectangle has a mosaic-like appearance that represents each category in an area that is proportional to the volume of references attached to that category.

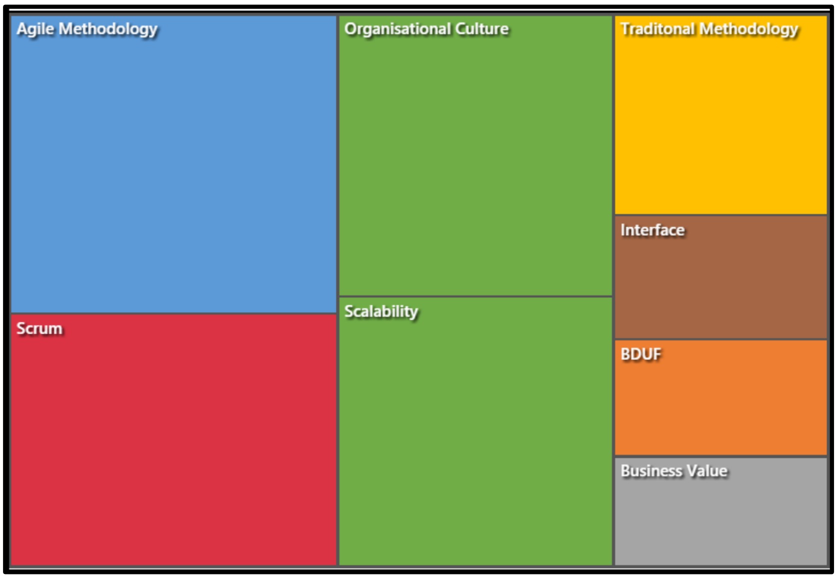


Figure 5.5: Hierarchical Chart Showing Volume of References

As shown in Figure 5.5, 8 categories received the most number of references. There were a substantial number of references attached to the categories of Agile Methodology, Scrum, Organisational Culture, Scalability, Traditional methodology as embodied by the Waterfall approach, Interface (from the UX (user experience) perspective as well as the functional interfacing with existing systems), Big Design Upfront (BDUF) and Business Value. The relatively large cohort of references to agile methodology was however, an expected outcome because much of the conversation with the interviewees had an agile focus. The references to organisational culture (OC) was also prescribed. However, the enthusiastic responses with regards to OC escalated the priority attached to OC so that it became one of the emergent themes in the analysis. The other main contributions such as the prominence of Scrum methodology, the constant reference to the Waterfall approach (coded as a sub-node of the main node named Traditional Methodology), the relevance of a BDUF strategy and the imperative for attainment of quick business value were all pivotal in providing the researcher with a focused stance for the subsequent interviews.

In order to provide better insight into the broad themes that were emerging, references to the raw data will be done by making use of "rich thick descriptions" (Creswell, 2013, p. 201) and verbatim excerpts from the interview transcripts. In accordance with the default agreement of confidentiality that was prescribed in the consent section of the interview schedule, the anonymity of the interviewees will be preserved by using a strategy whereby an identifier will be used to make reference to the interview number as well as the capacity that interviewees serve in their respective organisations.

The excerpt below is a comment regarding the scalability of software development methodology. This comment was made by an experienced agile practitioner who is currently employed in a software engineering capacity at a high profile software consultancy organisation in South Africa. This comment was made in response to an open ended question where the interviewee was asked to offer an opinion on software development methodology.

> *Many people have focused on the development aspect but tend to neglect how you run an agile project from a programme or portfolio perspective. Because systems are not built as stand-alone systems, they interface with other organisational systems. So I think that agile and Scrum methodology needs to become scalable so that it takes into account other organisational systems and ensures that agile teams work in synchronisation with other development teams as well as with testers and operations and quality control people.*
>
> **(Interview 4, Software Engineer)**

This comment became a catalyst for the researcher to explore issues related to the fitness for purpose of agile methodology, not just from a software development perspective but also from an organisational interfacing perspective.

The broad comments that were made with regards to general software development methodology as well as agile development practices were taking on a discernible pattern where references were made to business value and the need for extensive upfront planning so that the expected business value is not compromised. An excerpt that adds credibility to the preceding claim is referenced to a transcript of an interview with a respondent from the banking sector.

> *… the lack of time invested in upfront planning costs more in the end because there has to be a lot of rework that entails additional resources, costs and time…I think that it is better to fully specify all the requirements upfront because then we can scope the project and allocate the necessary resources which is all linked to the expected business value.* ***(Interview 3, Business/Systems Analyst)***

There seems to be a tendency for the IT practitioners to have a preference for the stability offered by extensive upfront planning where the project management issues as well as the business value aspect of a systems development effort is given the highest priority. Aligned to the business value and project management imperative is the significance of the operations component where the developers are expected to take ownership of these systems right up until the point of delivery, deployment and production of these systems into a live environment. The live environment view of these systems is where business value and investment in system resources can be justified and appreciated. The significance of this "live view" of the system and the significance of a DevOps approach is highlighted from the verbatim excerpt quoted below:

> *…so DevOps or shall I say the Ops part is crucial because they don't just look at a solution in an isolated way like the developers do, the Ops people have a global view of the value and relevance of a system, so in terms of the development effort and the use of agile or waterfall it all comes down to Ops people who then bring these systems to life and that is when you have a true idea of how well the system is working and enabling business value.* ***(Interview 3, Business/Systems Analyst)***

The initial data analysis was useful in enabling a better interview protocol because the prominent discussion points were noted and added as probes (see Appendix A for interview protocol) to enhance the depth and quality of subsequent interview sessions. New probes (questions used to elicit a deeper insight into a phenomenon) were added to the interview protocol based on the analysis of the 1st 9 interview transcripts as well as the memos that the researcher compiled at the end of each interview. These probes are listed below:

- The relevance of the *scalability* of software development methodology so that is has a strong alignment with the current IT infrastructure;
- The significance of *DevOps* (mentioned in all of the 1st 9 interviews) and how it enables the scalability of software development methods;

- The alignment of the methodology of development with organisational processes to enable a seamless acquisition of *business value;*
- The integration of processes into the methodology of software development to ensure that newly developed/ reworked systems *interface* seamlessly with the IT infrastructure at enterprise level. Incorporated as part of the interfacing dialog, the role played by issues pertaining to the human computing interface (HCI) will be examined. This includes coverage of aspects such as general usability and user experience (UX) design as well as user acceptance testing (UAT);
- The relevance and intensity attached to upfront planning sessions and the role played by a Sprint Zero (mentioned by 6 of the 9 interviewees).

Each interview was preceded by the administering of a set of Likert Scale questions (referred to as the pre-questionnaire in Appendix A). The responses to the Likert scale items were used as a catalyst to enhance the hermeneutic component of the semi-structured interview by enabling quick and early identification of aspects of ASDM that warranted further inquiry. The main outcomes of the pre-questionnaire phase are listed below.

- Scrum is the de facto methodology for software development. This response was made by all 16 interviewees;
- The Scrum and XP methods typically associated ASDM were all endorsed as useful. The exceptions being the use of pair programming and the presence of an on-site customer. Pair programming was not done with any formal rigour. The on-site customer was represented by the Product Owner (PO) or the Business Analyst (BA). Both these roles were used to subsume the role played by the onsite customer in systems development. The daily stand up meeting received emphatic endorsement by all of the

interviewees and became the source of further inquiry during the interview session;

- The use of systems design models followed a routine pattern of general acceptance of the main design models that were listed. From an analysis perspective, Data Flow Diagrams, User Stories and Use Case modelling were endorsed as important development models. The preference for lightweight analysis and design modelling was further explored during the main interview session;
- The issue of Big Design Upfront (BDUF) did generate varying responses and became the source of further inquiry during the main interview.

At a preliminary stage, based on the analysis of input from the 1st 9 interviews and the pre-questionnaire, a hypothetical disposition towards a description of the phenomenon of software development methodology as experienced by software development practitioners in South Africa is as follows:

- The **culture** within an organisation strongly influences the adoption of a software development methodology;
- The issue of **extensive upfront planning** has elicited varying opinions although the weight of evidence is gravitating towards the use of an extensive planning phase sometimes referred to as a **Sprint Zero;**
- Agile software development is highly endorsed but it needs to re-establish focus on the **business value** that the systems development effort provides; this should ideally be done at the planning phase of development which should receive more priority than is currently allocated by agile methodology;
- **Scrum is the de facto methodology** of choice for software development in South African based organisations; from a pure software development perspective, Scrum works well; Scrum is

supplemented with user stories and test driven development as well as the Scrum Board or the Kanban Storyboard to enable project management;

- The issue of interface plays a dual role. A reference to **interface** is contextualised as a reference to the user interface and a reference to the capacity of newly developed systems to interface with the organisational technical infrastructure;
- **Deployment** of newly developed systems needs to be given higher priority because a major stumbling block to agile methodology is that these systems only provide evidence of **business value** once they have demonstrated a propensity to interface with existing organisational systems in a live/production environment. **The use of a DevOps strategy** is endorsed as a viable mechanism of achieving good systems visibility from which business value can be quickly ascertained.

The purpose of this listing is aligned to the suggestion by Corbin and Strauss (2014, p. 375) that the earliest interviews must be analysed for "significant happenings" so that core concepts may be identified and used as a platform for further analysis. The core concepts identified (listed above) were used as a guiding framework to refine and reduce the extensive listing of 68 codes/nodes identified in the first phase of coding.

In order to establish the validity of the core concepts that have been identified thus far, a further set of 7 interview transcripts were added to the data corpus for analysis. The process of 1st Cycle coding was conducted on the latest data set. At this stage there was a total of 16 interview transcripts that were subjected to 1st Cycle coding. The coding process was becoming a routine exercise because the list of 1st Cycle codes was extensive and ensured that much of the text from the newly added sources was easily accommodated in the existing set of 1st Cycle codes. The exception was the addition of 3 new 1st Cycle codes bringing the total number of 1st Cycle codes to 71. A discussion of the 3 new codes that were added is warranted based on the relevance and importance of these new codes.

The **first new code** added is a reference to 'legacy systems'. Three of the interviewees made reference to the influence of legacy systems and agile software development in an organisational context in South Africa. It should be noted that the issue of legacy systems was mentioned during the first batch of transcriptions but the text accompanying this concept was coded under the node named *Waterfall* because in all cases, reference to legacy systems was accompanied by a reference to the use of Waterfall methodology in the development of these systems. A note was recorded by the researcher in the memo attached to the interview transcription so that reference to the narrative on legacy systems could be quickly retrieved if this concept begins to play a significant role (a strategy advocated by Corbin and Strauss (2014, p. 240)). The relevance of the memo was soon realised when the issue of legacy systems was given a bit more prominence in the 2nd batch of interviews after it was mentioned by 3 more interviewees. This was sufficient evidence to warrant the recording of legacy systems as an additional code. A retrospective coding exercise revealed significant statements that were made during the 1st batch of interviews but were not recorded as a significant contributor to the set of 1st cycle codes. This situation was rectified. An example of this situation is illustrated by the verbatim excerpt taken from the 1st interview attesting to the phenomenon of legacy systems. The 1st interviewee, who has 8 years of experience with agile software development, serves in the capacity as a software engineer and systems development manager at a government controlled national freight and logistics provider in South Africa.

> *You must understand that there are many legacy systems that still prevail with many South African organisations and these legacy systems are maintained using traditional systems development methodology. So even though organisations are making an effort to move to agile, there are pockets of waterfall mentality that still prevail in these organisations. (**Interview 1, Systems Development Manager)***

The **second new code** added is a reference to a 'Bimodal Approach'" to software development. This term is used in Horward (2015, p. 8) to advocate the Gartner Group's contribution to enabling the scalability of agile software development by suggesting that "you must be part solid and part fluid to thrive in digital business." The analogy is used to align Waterfall methodology with the concept of having a solid base/foundation and agile methodology with something that is quite fluid and not well established. The Bimodal approach entails the practice of implementing two separate but related styles of problem solving. Mode 1 is used for problems that are well understood and predictable and is ideally used for transitioning a legacy environment so that it is compatible with the requirements and standards of the current digital environment. Mode 2 is a reference to a more exploratory approach where the problem domain is highly unpredictable and requires a problem solving strategy that embraces uncertainty and has a high threshold for managing the impact of changing requirements. According to Horward (2015), a bimodal approach is required to drive organisational change that embraces the digital transformation. A catchy phrase used by Horward that links software development to this bimodal approach is that "*slow and steady* (a reference to Waterfall for the legacy systems) *plus fast and agile (*a reference to the methodology for newly developed systems*) wins the race.*" The bimodal approach is linked to the 'renovation' of legacy systems as suggested in the verbatim excerpt from an interview with the manager of the business architecture division at a private logistics provider in South Africa.

> *…there are many failures with agile implementation. It is because of the culture of adoption and the resistance to change and it is not ideal when transitioning legacy systems. Our approach is to use a Bimodal strategy where there is a slow transition. The smaller teams working on smaller projects that are focused on establishing competitive advantage should work in an agile way. The bigger teams that are doing more critical development projects that are invariably linked to the mainframe legacy systems that actually run the core processes of the company should continue in a traditional Waterfall way.*

(Interview 8, Senior Business Architect)

These comments suggest a strong link between the dilemma of handling the upgrade of legacy systems and the gravitation towards agile methodology in South Africa. The transition to agile methodology is also inextricably linked to an organisational cultural shift that cannot be implemented instantaneously. There is also a legacy mentality that needs to be transitioned in a controlled and pedantic manner as is suggested in the verbatim excerpt taken from Interview 8 that links up the complexities of handling a transition to agile methodology in a very eloquent manner.

> *The reason for this bimodal approach is that the required functionality of the legacy systems is highly predictable and development can be done in a more structured way. Also, it enables the traditionalists who have been with the organisation for many years to continue working in their comfort zone. However, as the smaller teams become more accustomed to the workings of agile and as they enjoy success, then there should be some sort of knowledge transfer to the bigger teams. You see in order to enable a culture of transformation in an organisation you have to demonstrate the success of the new methodology and you must remember that agile is not a recipe for success.*
>
> **(Interview 8, Senior Business Architect)**

These comments are pivotal in addressing the influence of organisational culture which has been identified in the literature as a major stumbling block in the transition to agile methodology. From an academic perspective, the reference to "traditionalists" is a reference to the Group Culture dimension that was part of the academic discourse on organisational culture and the Competing Values Framework presented in the literature review. The Bimodal approach also resonates with a "phasing in" approach that was mentioned in Interview 1.

> *Recently I've noticed a small shift towards a more relaxed stance towards development but this happening very slowly. The terminology that we are using now is that we want to "cannabalise" our existing traditional methodologies. The idea here is that we slowly eliminate aspects of Waterfall methodology until there is nothing left. This approach is good because the senior guys feel very much in control of the transition and it is not just a big bang approach where everything*

> *is moved to agile immediately. So we're basically phasing in the agile approach and phasing out the Waterfall approach.*
>
> **(Interview 1, Systems Development Manager)**

A review of the original set of codes revealed that a classification named *Transition from Waterfall* had already been created and there were 8 references from the 16 interviewees to this category. An examination of these references revealed that the transition from a Waterfall approach has been described by the interviewees through the use of terminology such *phasing-in* or a *parallel transition to agile methodology*. The proximity of meaning between these terms and the formal use of a strategy such as the Bimodal Approach was noted as an area for refinement of codes during the 2nd cycle coding phase.

The **third newly added code** was security. Once security was added as a new code, retrospective coding had to be done just to ensure that implicit references to security oriented issues were not missed during the first coding cycle. The role played by issues pertaining to security of systems has become prominent with modern information systems. From a development perspective, security has become an integral part of the testing phase. However, this casual arrangement warrants more focus as evidenced by the following verbatim extract:

> *…the main thing about operations is to ensure that the systems fit in quite well with the enterprise infrastructure...it must not cause problems with other systems but it must also not compromise the security of these systems as well because the security concern is huge in the banking domain. Sometimes there is a bit of confusion about where the security people are meant to function ...with the developers or with the operations people but there has to be explicit focus somewhere in the process on security management.* **(Interview 14, Operations/DevOps Team Leader***)*

The initial set of 71 codes served as a foundation for further data analysis that was conducted as part of the 2nd coding cycle aligned to the qualitative data analysis methodology suggested in Saldana (2009). However, prior to 2nd Cycle coding many of these 71 codes were re-configured and conflated into existing codes so that the initial set of codes could be reduced to a manageable set.

The refinement of codes was done strategically so that references to rich textural descriptions of the experience of the phenomenon of agile methodology obtained from the interviewees was not lost. The refinement was done in accordance with suggestions in Saldana (2009) as well as Bazeley and Jackson (2013). The strategy entailed:

- Merging or removing responses to probes/questions that alluded to concepts that did not elicit much emotion or the interviewees were of the opinion that these concepts were not an area that would enable any improvement in the experience of software development;
- Deletion of codes that were not deemed to contribute to the envisaged outcome of the current study were deleted;
- A hierarchical arrangement of codes/nodes to enable better comprehension of the underlying data.

The outcome of this exercise was to refine the initial set of 71 first cycle codes into a more manageable set of 40 codes which were then subjected to further refinement during the 2nd Cycle coding phase described in the subsequent section.

5.3.3 Second Cycle Coding

According to Saldana (2009, p. 149), the main objective of Second Cycle coding is to "…develop a sense of categorical, thematic, conceptual, and/or theoretical organization from your array of First Cycle Codes". Basically this phase of coding entails a reconfiguration of the 1st Cycle codes in order to develop a finer set of broader codes that each encapsulate chunks of 1st Cycle codes. Saldana (p. 150) identifies 6 techniques for 2nd Cycle coding. These techniques are not mutually exclusive in their methodology and Saldana suggests that they may be combined in a "mix and match" arrangement that is referred to as "Eclectic Coding".

For the purposes of the current study, 3 of the 2nd Cycle coding techniques were identified to enable further condensation of the data corpus. These are:

- Pattern Codes – used to categorise similarly coded data and is sometimes referred to as a meta-code based on conceptual similarity also referred to as “clustering” in Huberman *et al.* (2013, p. 279);
- Focused Coding – used to streamline the set of First Cycle codes by establishing focus on codes that occur most frequently and is perceived as being the most appropriate in the context of the research questions. It also enables convergence of the diverse set of First Cycle Codes so that further analysis is directed towards obtaining a cogent understanding of the main phenomenon of the study. In the context of the current study, this method of coding was used to remove codes that represented some of the periphery aspects related to the phenomenon of software development as experienced by practitioners in South Africa;
- Axial Coding – regarded as an extension of initial coding/First Cycle coding and entails the reassembling of data that was “fractured” during the First Cycle coding phase. This form of coding entails identification of a central word/term that serves as an appropriate descriptor for the text that is coded under this “axis”.

The process of 2nd Cycle coding, informed by the coding methods listed above, has been implemented on the set of 1st Cycle nodes to reveal a coding configuration that consists of 12 Second Cycle codes. The salient aspects of the code reconfiguration exercise from 1st Cycle into 2nd Cycle codes is explained in Table 5.1.

Table 5.1: Transition from First Cycle to 2nd Cycle Coding

<table>
<tr><th></th><th>1st
Cycle Code</th><th>Action Taken</th><th>Reduction Technique</th><th>2nd Cycle Code</th></tr>
<tr><td></td><td>Management Control</td><td>Added as a child node to code named <em>Agile Methodology</em></td><td>Axial Coding</td><td>Agile Methodology</td></tr>
<tr><td>Researcher Reasoning</td><td colspan="4">The direct influence of management control over agile teams seems be on the decline in the context of the migration to Agile Methodology; the transition to agile is accompanied by a more democratic style of management where all team members take joint responsibility for the systems development effort.</td></tr>
<tr><td>Supporting Evidence</td><td colspan="4"><em>Well I haven't reported to a project manager for some time now. Although we still perform project management, but this is done so that we can track the project's progress. Our main role players are the product owners, developers, testers, BA's and possibly a scrum master although there are times when we all assume that responsibility. Also, I feel that if we want to move away from the Waterfall mentality where everything was command and control and management driven then this shift in thinking is actually a good one. <strong>(Interview 9, Software Developer in banking sector)</strong></em>
<em>We are basically given a system to develop and once the team is formed, we identify our product backlog together with the Product Owner and then we start with the Scrum development cycle. We do a lot of workflow tracking by making use of a combination of the Kanban Board and the Scrum Board. We normally just create these progress charts on a white-board as you see all around us. We make use of colour coded stickers to track progress. This approach is easy to manage and works well for us because it gives us a good sense of knowing the status of our development efforts. Also we are able to identify bottlenecks quite easily and handle them in our standups. We pretty much manage ourselves in this way. <strong>(Interview 16, Analyst/developer for a national software organisation)</strong></em></td></tr>
<tr><th></th><th>1st
Cycle Code</th><th>Action Taken</th><th>Reduction Technique</th><th>2nd Cycle Code</th></tr>
<tr><td></td><td>Big Design Upfront (BDUF)</td><td>Merged into existing Code named Planning</td><td>Pattern Coding</td><td>Planning</td></tr>
</table>

Researcher Reasoning	The issue of BDUF elicited highly emotive responses ranging from a ‘compulsory requirement’ to an ‘optional requirement’ that depends on the organisational imperative as well as the type of project. The general consensus was that the issue of BDUF should be deliberated upon at the planning phase and there is a preference for lightweight design models.
Supporting Evidence	*…this is an aspect where I think that Waterfall may have had its advantage over Agile because the initial focus on planning and upfront design at the requirements phase basically got everyone on the same page, whereas with agile, you sometimes have not defined the requirements in full and you end with a system that does not deliver explicit business value or does not fully meet the business requirements that were expected of the system. But I think that Agile has removed much of the intensity of modelling and this is a huge improvement on Waterfall because the system starts unfolding quite quickly so that if there are design flaws, you pick it up early.* ***(Interview 3, Business/Systems Analyst in banking sector)*** *…we are trying to do as much upfront analysis as we can in order to try and cost the project as accurately as possible. There can be no grey areas in this regard because our company is a third party vendor, we have to be very clear with the client upfront with the system features that will be developed. The primary purpose here is that the client can be accurately billed for the effort that it takes to develop the system. From a requirements perspective, we do this a little more dynamically where we use the upfront design as the baseline set of requirements and then adjust and modify as we consult with the client regarding incremental features that are added on.* ***(Interview 2, Manager at a Software Development Organisation)*** *I’m trying to push for a slight drop in the amount of upfront analysis and design that we currently do. Right now we are sitting at 100% requirement that all the analysis and design is completed, inspected, signed off, and only then do we invoke our development teams into a scrum based arrangement. But the upfront planning is non-negotiable and thus far we do not trust the developers to allow the system design to evolve with the coding.* ***(Interview 11, Chief Software Development Methodologist in the banking sector)***

	1st Cycle Codes	**Action Taken**	**Reduction Technique**	**2nd Cycle Code**
	Documentation, maintenance and error handling	Merged into existing Code named *Quality Control*	Axial Coding & Pattern Coding	Quality Control
Researcher Reasoning	The importance and relevance of documentation and maintenance and error handling was often mentioned in the context of quality control;			
Supporting Evidence	*So in a perfect world where everyone is focused on a single project, the documentation is not that crucial because there is a strong focus on the one project. But when you are working on multiple projects concurrently then good documentation is crucial to ensure that you build quality systems…this makes maintenance a lot easier because errors or issues picked up later in the development cycle can be handled a lot quicker and easier. This becomes a problem when developers who use agility as an excuse to reduce documentation because it causes a huge time delay to re-establish the context of the requirements as well to track progress of the project. We call it re-harvesting the design and documentation and this is time consuming.* ***(Interview 3, Business/Systems Analyst in banking sector)***			
	1st Cycle Code	**Action Taken**	**Reduction Technique**	**2nd Cycle Code**
	Ownership & System Visibility	Merged into existing Code named *Scrum*	Axial Coding	Scrum
Researcher Reasoning	The issues of ownership and system visibility were identified as the main contributors to the popularity of Scrum methodology in South Africa.			
Supporting Evidence	*…one of the other things that I find with scrum…the developers bind to the idea of the development schedule, they bind to the idea and expectation of the daily stand ups, they bind to the idea that we're releasing a sprint and they adopt an attitude that says at this stage, everything has to be in order …this promotes good work ethic amongst the developers because they know the deadlines and the expectations on a daily basis so Scrum ensures that there is a sense of ownership because things around a specific project are*			

	happening on a daily basis. **(Interview 2, Manager at a Software Development Organisation)** *From my experience, scrum and agile provide a better turn-around time…customers get to see the system a lot quicker…the development team has better ownership of agile systems because once you get customer or product owner feedback then there is a natural urge for you to want to act on that feedback and then go back to the customer and say is this what you were talking about...there is an immediate sense of knowing that we're built the right thing.* **(Interview 7, Software Developer in banking sector)**			
	1st Cycle Code	**Action Taken**	**Reduction Technique**	**2nd Cycle Code**
	DevOps & SAFe	Merged into existing Code named *Scalability*	Axial Coding	Scalability
Researcher Reasoning	There were numerous references to the DevOps approach that many organisations in South Africa are beginning to adopt. However, there have been 2 instances where reference has been made to the Scaled Agile Framework (SAFe) which represents a highly mature version of the DevOps approach. The DevOps/SAFe ideology is encapsulated under the name Scalability. The reasoning here is that both these strategies are aligned to the imperative to ensure that an agile mentality becomes ubiquitous throughout the organisation			
Supporting Evidence	*…the DevOps initiative enables better communication and this is what breaks down the silos that previously existed. In the past you had your Agile teams and there was still very much a Silo mentality. Now with the DevOps mentality, everyone works together from the business architects, the planners, the developers, testers, the network administrators and because everyone works together communication is not such a big problem anymore and agile is scaling up to an organisational level… the only problem that I've seen with other banks as well, is that everyone wants to go straight to DevOps and there is this talk about SAFe but I think it's more of a journey. People wan'na run before they can walk. It's the small things people need to start looking at such as easy to digest changes where we can break the silos.* **(Interview 9, Systems Architect in the banking sector)** *So the empowerment of agile teams from a business perspective is hugely important and I suppose the DevOps teams feel a lot more empowered than the pure Scrum based or agile teams and so with DevOps there is better ownership of the product and the team members feel empowered to protect and maintain their system*			

	because they can see the business value at company level and feel that they are responsible for generating this business value. **Interview 4, Solutions Architect at an International software consultancy** *)*			
	1st Cycle Code(s)	**Action Taken**	**Reduction Technique**	**2nd Cycle Code**
	Wagile and Bimodal	Added to existing Code named Transition from Waterfall	Axial Coding & Pattern Coding	Transition from Waterfall
Researcher Reasoning	The transition from Waterfall methodology has been done either as a phased-in approach, a Wagile approach or a bimodal approach. This transition is epitomised by aspects of agility and Waterfall being integrated into a customised version of agility.			
Supporting Evidence	*But I've noticed that there is a lot of Waterfall stuff that goes on in Scrum based teams. So each Sprint consists of a Waterfall approach and the entire Scrum based project is driven by a Waterfall plan because we may be adding functionality incrementally, but this is not released. We release the system when all the specified functionality is available, the system has been intensively unit and integration tested and then we hand over the system for release. So in a way you can say that we think we are agile but we're actually 'wagiling' the whole thing.* ***(Interview 3, Business/Systems Analyst)***			

The initial set of codes contained references to the technical aspects of software development such as object orientation, code reusability, UML design models, inheritance hierarchies as well as strategies such as pair programming and code refactoring. By making use of a focused coding strategy, the textual content classified under these codes were either conflated into codes that had more relevance to the research objectives or were eliminated altogether.

Once the process of 2^{nd} Cycle code refinement had been completed, there was a total of 12 Second Cycle codes/nodes that remained. While this is quite a substantial reduction from the initial set of 71 First Cycle codes, many of the 1^{st} Cycle codes were clustered into the 2^{nd} Cycle coding categories as child nodes so that the richness of the verbatim responses were preserved. A frequency graph displaying the 2nd cycle nodes as weighted percentages is illustrated in Figure 5.6. Included in Figure 5.6 is a frequency count of the number of interviewees who 'contributed' towards the frequency value of each of the 12 nodes.

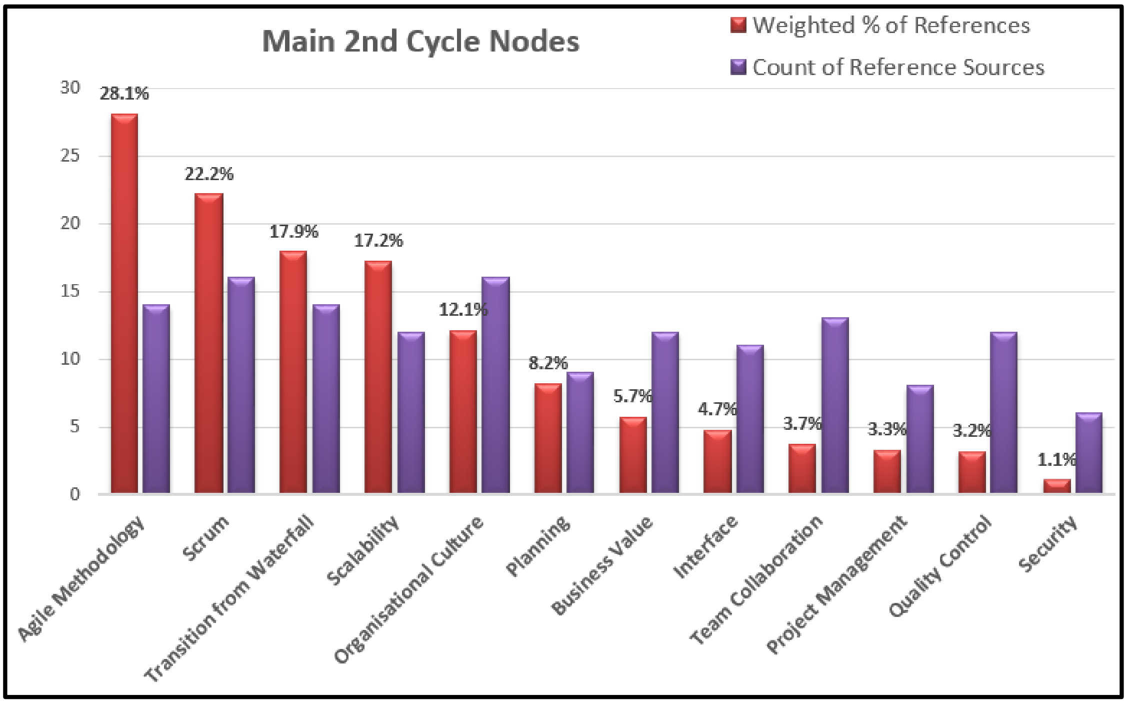


Figure 5.6 A Frequency illustration of the 12 2nd Cycle Codes

The data coded under each of the main nodes illustrated in Figure 5.6 is composed of sub nodes that are used to add structure to the data. A sample of this hierarchical structuring is illustrated in Figure 5.7

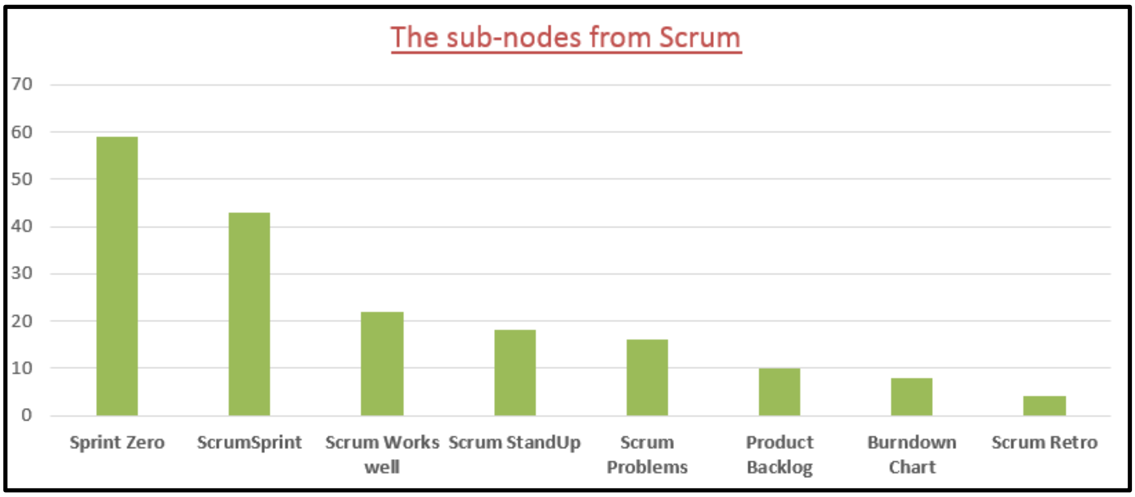


Figure 5.7: Frequency of References to the Scrum Node and its Child Nodes

The *Scrum* node contains 8 sub-nodes (child nodes). Figure 5.7 illustrates a frequency count of the general coding references to the 8 Scrum child nodes. The conflation of 1st Cycle Codes into 2nd Cycle codes was done mechanically by making predominant use of the Axial and Pattern coding techniques which worked quite well for many of the codes that could be clustered together quite naturally. The outcome of the strategy is illustrated in Figure 5.6 with reference to the 2nd Cycle Scrum node that was created by grouping together many of the concepts that form an intrinsic part of Scrum methodology. However, the main Scrum node also contained references to concepts/methods that are not naturally associated with Scrum methodology. These include test driven development (TDD) (an intrinsic part of XP), design models (an intrinsic part of the software development process in general and not necessarily a Scrum based concept), system ownership, system visibility and team collaboration (abstract concepts that allude to general traits of the software development process). These concepts/methods were clustered as part of the main Scrum node because a review of the context in which these concepts were mentioned by the interviewees revealed that in all cases, it was part of the discussion of Scrum methodology.

The frequency counts that were undertaken such as the illustration in Figure 5.7 for the Scrum node provided the researcher with a quick overview of the patterns that were emerging from the data. In the case of the Scrum node, a discernable trend is the interviewees' gravitation towards the opinion that Scrum is regarded as the methodology of choice for software development and has become a de-facto standard in this regard. This assertion is based on the observation that in all 16 interview transcripts, Scrum was assumed to be the default methodology used for software development. The only other discernable methodology that was mentioned was the Waterfall methodology. However, this was more in the context of an approach to software development that needed to be changed because of a legacy mindset. The expected outcome of the envisioned change was to embrace agile development by adopting a scrum/scrum-like approach. As illustrated in Figure 5.6, a relatively high percentage of the interview transcripts were coded as Scrum (22.2%, n=16) as well as Transition from Waterfall (17.9%, n=14).

By running a query of the number of joint references to *Scrum* and *Transition from Waterfall*, it can be observed in Figure 5.8 that there were 14 sources that made a reference to the *Transition from Waterfall*, and all 14 also made a reference (in the same paragraph taken from the transcript) to *Scrum* Methodology. The majority of the coding references (as illustrated in Figure 5.8) to Scrum also included a reference to the Transition from Waterfall (and vice versa).

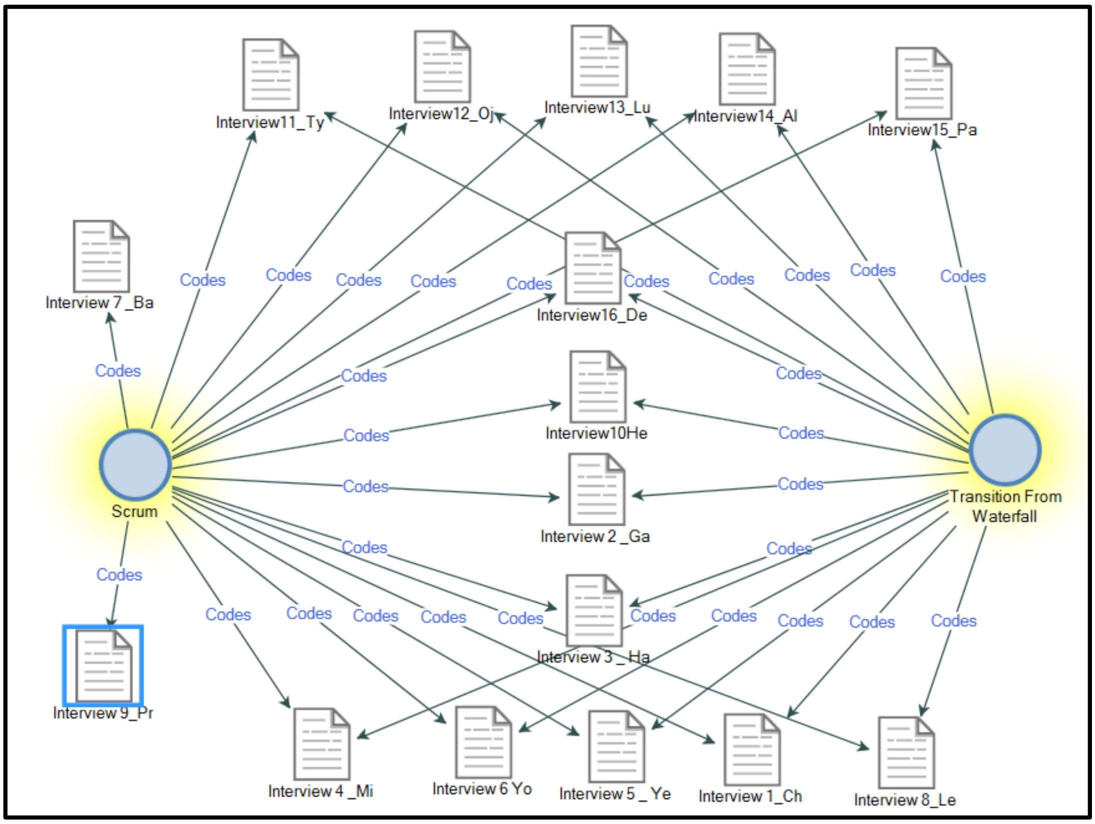


Figure 5.8: Cross Query between Scrum and Transition from Waterfall

Only 2 of the transcripts made coded contributions to the Scrum node and did not make a contribution to the Transition from Waterfall node. Upon closer examination of both these transcripts, it was established that both the interviewees were software developers who were currently working on Scrum based projects and the transition from Waterfall was not perceived as an issue because Scrum methodology had been prescribed at the organisational level. In the case of the 14 interviewees who made concurrent references to Transition from Waterfall Methodology and Scrum Methodology, a plausible inference that could be made is that the transition from Waterfall Methodology culminates in the adoption of Scrum as the preferred methodology for software development.

Another discernable relationship that can be established from the analysis of 2nd Cycle codes is the issue of scalability and the imperative to build mechanisms of scalability into the Scrum development methodology. The use of the term scalability is in reference to the potential for systems that have been developed using agile /Scrum methodology, to be in a 'production-like' state or for the system to acquire a capacity to seamlessly integrate with enterprise-wide systems. This concern has manifested in references to the node named Scalability which includes DevOps as a child/sub-node. The relationship between Scrum and Scalability is illustrated in Figure 5.9.

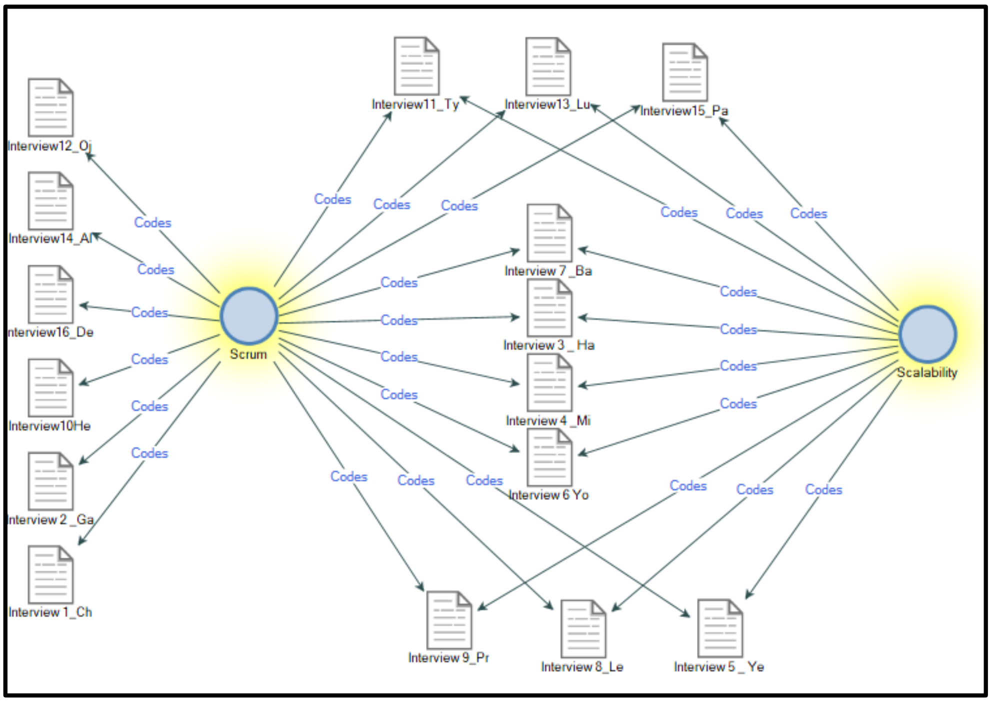


Figure 5.9: Cross Query between Scrum and Scalability

From Figure 5.9, it can be established that there were coded contributions from all 16 of the interviewees to the Scrum node. In 10 of the 16 (62.5%) interviewees made a reference to the nodes Scrum and Scalability in the same context. According to Huang (2008, p. 49) and Niwattanakul *et al.* (2013) a cluster analysis test is a good indicator of the similarity of word content between textual documents. The Jaccard coefficient may be used as an indicator of the word similarity. The cluster analysis between the nodes Scrum and Scalability revealed a Jaccard coefficient of 0.76 indicating a 76% similarity index that made reference to Scrum and Scalability in the same context reveals a pattern that attests to the inability of Scrum Methodology to provide solutions that are easily scalable to an organisational platform.

A further query based analysis of the relationship between the nodes Scalability and Business Value revealed the structure illustrated in Figure 5.10. Eight of the coded references (50%) to the node Scalability also included references to the node Business Value in the same context.

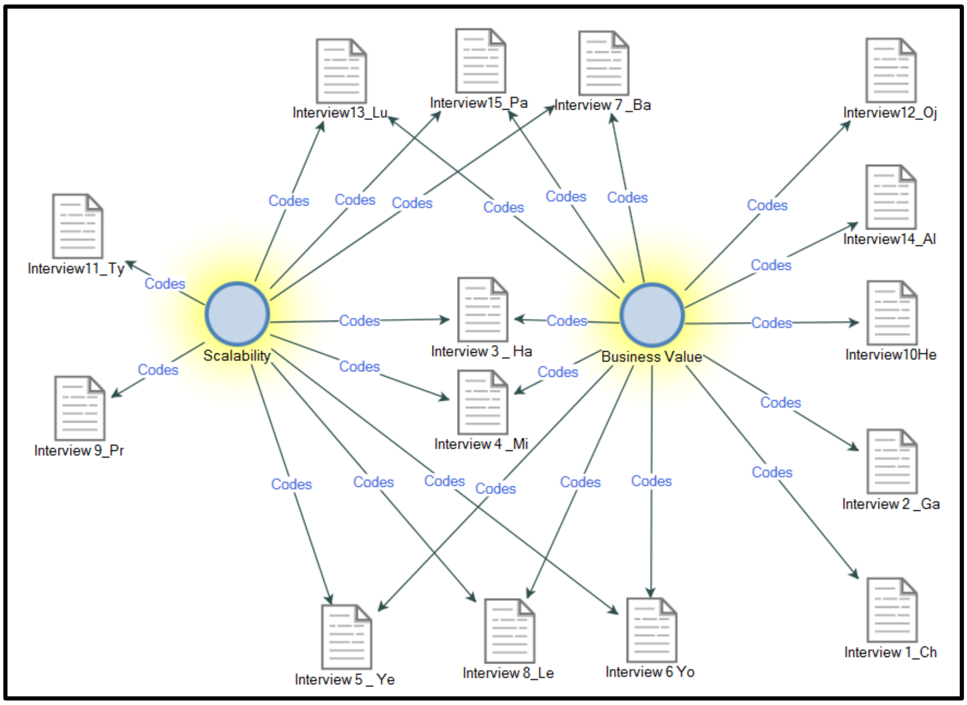


Figure 5.10: Cross Query between Scalability and Business Value

As can be seen in figures 5.9 and 5.10, there is a significant amount of commonality between codes attached to the nodes Scrum, Scalability and Business Value. Emerging patterns such as this provides the researcher with an opportunity to verify this outcome in subsequent interviews and also enables an informed convergence towards a holistic and summative representation of the data corpus that is accurate and complete. The quantification exercise consisting of frequency count illustrations, the cluster analysis as well as the intersection queries is being used to obtain overview knowledge of the data corpus. In keeping with the dictates of qualitative analysis, the significance of the actual numeric values attached to the frequency count for the nodes/child nodes do not have a bearing on the outcome of the analysis or as Huberman *et al.* (2013, p. 287) points out, "*…numbers tend to get ignored*". However, there is an assertion that qualitative researchers have a reliance on tactics such as frequency counts and word/term correlations in order to aid in the identification of themes or patterns. According to Huberman *et al.* (p. 288) the numeric option enables a qualitative researcher to obtain a quick overview of the main components of a large batch of data, provides support for the verification of a hunch or hypothesis and "*…to keep yourself analytically honest, protecting against bias*".

5.3.4 Rising above the Codes

The analytical phase of the qualitative data analysis culminates in the identification of several categories, major themes or concepts or possibly at least a theory that provides the highest level of abstraction to the underlying data corpus (Saldana, 2009). This is usually achieved after the process of 2nd Cycle Coding.

This synthesis phase of qualitative research is however highly interpretive and there are many diverse methods that may be used to "crystalise the analytical work undertaken thus far" (Huberman *et al.*, 2013, p. 277). All of the methods have a common objective of systematically assembling a coherent understanding of the data.

Huberman *et al.* (2013, p. 290) make reference to a "logical chain of evidence" where individual data items are conflated into themes/conceptual bins to create a "*more economic whole*" that represents more than just the sum of its parts. A viable strategy to develop this "economic whole" is to make use of Saldana's (p. 187) *code weaving* method, which entails an integration of key code words and phrases into a narrative format in order to see how "the pieces fit together". The operational aspects of code weaving are to make use of 2nd Cycle

Codes to form the salient components of a narrative that provides a “story line” that may be used as a foundation for the development of a theoretical model (that could lead to a successful experience of the main phenomenon of the study). This process is eloquently described by Huberman *et al.* (2013, p. 292) as “…moving up progressively from the empirical trenches to a more conceptual overview of the landscape*”*. The act of putting together the narrative is dependent on the 2nd Cycle codes as well as the intuition and insight of the researcher. The latter is referred to as the “inferential glue” (based on the insight obtained by the researcher as a consequence of engagement with the data corpus) that binds the main emerging concepts into an overarching story line that accounts for the “how” and “why” of the phenomenon under study.

5.4 A Rich Textural Description of Software Development in South Africa

An outcome of a phenomenological study is to provide a rich textural description of the main phenomenon of the study as experienced by the subjects of the study. In order to present such a description, a code weaving narrative is used. The narrative is compiled by making strategic use of the main codes that underpinned the analysis phase of the study.

The rich textural description of the phenomenon of software development as experienced by practitioners in South Africa, based on the empirical evidence provided in this study (illustrated in Figure 5.6) coupled with the researcher’s interpretive analysis of the evidence is as follows:

- **Organisational Culture** has a substantial influence on the adoption of a specific software development methodology. The design adopted in the study necessitated a reference to specific types of **organisational culture** that could influence the adoption of software development methodology. An outcome of this heuristic approach is that there is a resonance between Developmental and Rational Culture and the practice of agile software development. The Hierarchical and Group Culture classifications were found to resonate more with a Waterfall-like approach to software development where there is a preference for strong management control. An appropriate excerpt that defines the role of culture in an organisation is provided in the following 2 verbatim excerpts.

> *The company is maybe hierarchical as you have defined it because management delegates tasks and responsibilities; but hereafter it is quite developmental and I suppose rational in the sense that teams manage themselves in an agile way but must account for time spent by demonstrating progress made towards meeting a client's requirements.* ***(Interview 2, Manager at a Software Development Organisation)***

> *I don't think that there is a culture that is consistent across the whole business, it is the culture within that particular team. We have got different dynamics, obviously we have corporate culture and that is hierarchical but then we have a vibe underneath this which I would say is quite developmental and agile.* ***(Interview 11, Chief Software Development Methodologist in the banking sector)***

A cross referencing analysis was compiled by using a frequency count of words that alluded to a specific type of organisational culture (OC) and its alignment to either an Agile Methodology (AM) or a Waterfall Methodology. The outcome of the cross referencing frequency analysis exercise is illustrated in Figure 5.11.

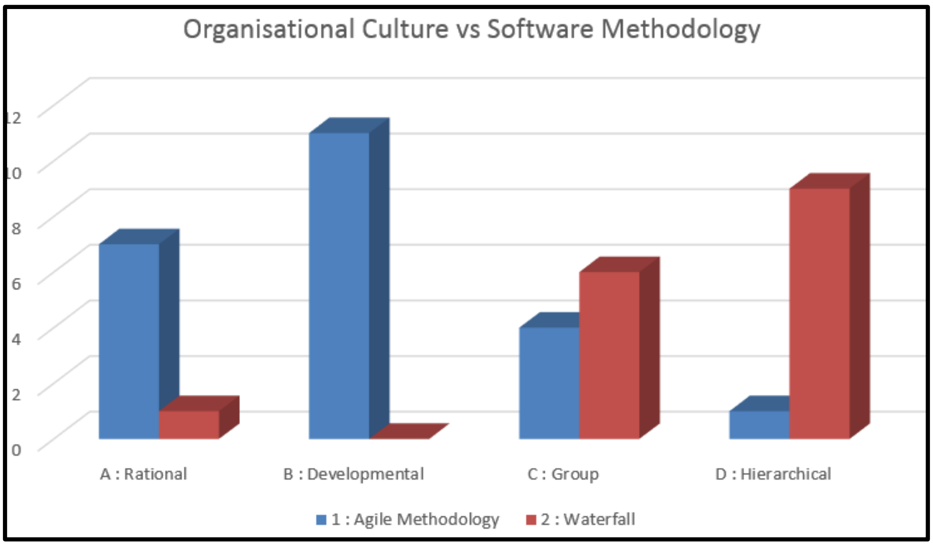


Figure 5.11: Cross Query between OC and Software Development Methodology

As can be established in Figure 5.11, there were 7 joint references to Rational Culture and AM and 1 joint reference to Rational Culture and Waterfall Methodology. Also, there were 11 joint references to the Developmental Culture and 0 references to Waterfall Methodology. The number of joint references to AM and Group Culture and Waterfall Methodology and Group Culture were similar. However, the Group Culture orientation seems to have a slightly stronger association with Waterfall Methodology. Another extreme case manifests in a high association between Hierarchical Culture and Waterfall Methodology;

- **Waterfall methodology** has a prominent role to play because it is seen as a baseline methodology that is used to contextualise deliberations regarding software development methodology. Also, many organisations in South Africa have core business reliance on legacy systems that have been developed and are currently maintained via a Waterfall approach. However, the overriding imperative of most organisations is to fast-track the **transition from the Waterfall methodology** towards an agile approach. The reason for the criticality of this transition is that the fast pace of modern business necessitates the use of innovation to arguably ensure that commercial organisations remain competitive. The software development approach that enables technological innovation has to prioritise speed of development to demonstrate **business value** and to respond to changing user requirements that occur because of the changing business landscape. This dynamic business environment is not compatible with a software development approach that abides by the dictates of Waterfall methodology;
- **The transition from the Waterfall methodology for software development** to an agile approach is currently being achieved via a phasing-in approach or a bimodal approach. The reason for

adopting a controlled and planned migration to agility is that there is a strong allegiance to the Waterfall approach by traditionalists within an organisation. The traditionalists, who may be classified as contributors towards a Hierarchical or Group culture, have a preference for strict management control that entails the upkeep of bureaucratic processes such as the "signing-off" of different phases of development by management before the next phase can proceed. The preservation of the Hierarchical and Group cultural traits within an organisation poses the biggest impediment to the transition to an agile approach to software development;

- **Scrum** has been endorsed as the **de facto strategy** for software development. The main benefit of Scrum is that it engages the development team into an intensive cycle of development that delivers evidence of **business value** in a short time frame thereby guaranteeing quick system visibility. Scrum is however not seen as a comprehensive methodology but rather as a constellation of methods that contribute towards successful software development. Scrum has been **hybridised** to include methods, design models and strategies that are prominently associated with methodologies such as Waterfall, XP and Kanban. The most prominent of these are the use of user stories (XP), data flow and entity relationship diagrams (associated with design phase of the Waterfall Methodology) and the KanBan Storyboard (used to track system development progress);
- **Scrum** has been flagged as being **problematic** from a **scalability** perspective. It provides an ideal framework for software development 'in the small' but does not 'scale-up' to the organisational level. A consequence of this dilemma is that software operations and configuration verbiage such as 'software release' and 'software deployment' have become intrinsically linked to software development practice. This situation has resulted in an **'infusion' of operations and configuration methodology into agile**

software development methodology. This heralds an extrapolation of the concept of agility to all spheres of an IT department in order for agile software development practice to reach its' envisioned levels of success;

- The issue of **scalability has been addressed via the DevOps initiative** which has become an area of much deliberation in the software development domain in South Africa. The reason for this focus on development and operations is that the operations aspect of software development has largely been ignored. This has resulted in a fracturing of association between software development teams and teams that have been designated as deployment teams. The deployment teams assume a DevOps role and are responsible for ensuring that newly developed applications are integrated into the organisational infrastructure. The current organisational imperative is to ensure that there is a quick turnaround time from the inception of a newly commissioned application to the release of that application into a 'live' production environment. This approach will arguably ensure that the newly developed application delivers the envisioned **business value** in a short development timeframe, thereby enabling the organisation's competitiveness and business viability. There is also an imperative to integrate a DevOps dimension into Scrum development as intimated by the following verbatim extracts:

  …the DevOps approach is quite new and I think it is good if you engage with the operations people early while you are in your Scrum mode. Because then the ops guys can configure your dev environment so that it matches the production environment and when you commit code you can quickly see your contribution to the main system and you feel a sense of ownership of the system because of your coding contribution. In the past there were always issues with the dev and production environment and during lunchtime talk...that's when you hear about how your code created some regression error...but if you are part of the

> *process the whole way then you feel a lot more in control (***Interview 16, Analyst/developer for a national software organisation)**
>
> *…just like how agile methodology enables system visibility from a technical perspective DevOps enables the visibility of business value of an IT application from a business manager's perspective*
>
> **(Interview 6, Software Engineer in the Banking Sector)**
>
> *…let's suppose you were able to see the business case unfold from planning to development to installation in a seamless continuous manner by removing all the bottlenecks…and these usually occur at the deployment phase…then I think that will be the ideal working arrangement because the BA is able observe and appreciate the business value quite quickly* **(Interview 6, Software Project Manager in the Petro-Chemical Sector);**

- There is still a huge focus on the **planning** (specification of business value, requirements specifications, analysis and design) as evidenced by the following verbatim extract.

> *… there tends to be a culture of mistrust and developers and managers are more comfortable when the crucial functional components are built exactly to specifications and all stakeholders know everything about the component and are able to predict paths of execution that may lead to regression errors. So the upfront specification phase is non-negotiable.*
>
> **(Interview 1, Systems Development Manager)**

The planning phase needs to incorporate strategies for **quality control** such as system **performance testing** and **user acceptance testing**. The agile philosophy of allowing the system design to evolve with the coding of the system is not readily endorsed. This assertion is based on the volume of codes that were recorded as part of the planning phase (Figure 5.6) as well as the number of references to a Sprint Zero (Figure 5.7) which is essentially a planning phase. An underlying driver for software process improvement is the imperative to ensure that software development efforts have a strong alignment to the delivery of **business value**.

This quest to achieve this alignment will ensure an organisational cultural-transition from what is perceived to be Hierarchical and Group culture to a culture that is more Rational and Developmental. The preceding assertion is made in the context of South African organisations.

5.5 A Proposed Model for the Adoption of a Software Development Methodology

The rich textural narrative presented in the previous section represents a synthesis of the diverse strands of empirical data encountered in the study. The status assumed by this narrative may be seen as descriptive in a static sense but can also be viewed as rather dynamic in an inferential sense. The narrative which is based on the experience of software development in South Africa by expert practitioners has an 'inevitable' theme of proposing a best practices guide/framework to arguably ensure future success in the use of software development methodology in South Africa. This assertion is made on the basis of the researcher's engagement with these practitioners who were driven by an innate desire ("…people are meaning finders" (Huberman *et al.*, 2013, p. 277)) not to simply describe their experience in the use of software development methodology but to also make suggestions that will contribute to an improvement of the software development process. In order to propose a framework that will enhance the prospect of the successful implementation of software development methodology in South Africa, a strategy of identifying the main themes or "gestalts that pull together many separate pieces of data" (Huberman *et al.*, 2013, p. 277) is used. This strategy is informed by the method proposed in Saldaña (2015, p. 187) that entails an identification of the **three main ideas** that emanate from the rich textural narrative of the main phenomenon of the study. Saldana calls this the **study's *trinity***.

The three main ideas/themes that binds the data corpus is listed below:

1. The culture of an organisation influences the adoption of the methodology used for software development. There is an imperative for South African business organisations to migrate from a Waterfall approach to an agile approach for software development. This can only be achieved by an accompanying shift in the culture within the organisation to one that is Developmental or Rational.
2. Agile software development has migrated towards a Scrum based approach. However, the proviso is that the Scrum approach is preceded by upfront planning sometimes referred to as a Sprint 0.
3. Software process improvement (SPI) efforts are driven by a desire to obtain discernable business value. There is a requirement to ensure that SPI strategies enable the attainment of business value by ensuring that the methodology adopted is highly scalable to the enterprise operating environment. This can only be achieved by factoring in the operational requirements of a system at an early stage of development.

The illustration in Figure 5.12 shows the core ideas that have emanated from the empirical evidence gathered thus far.

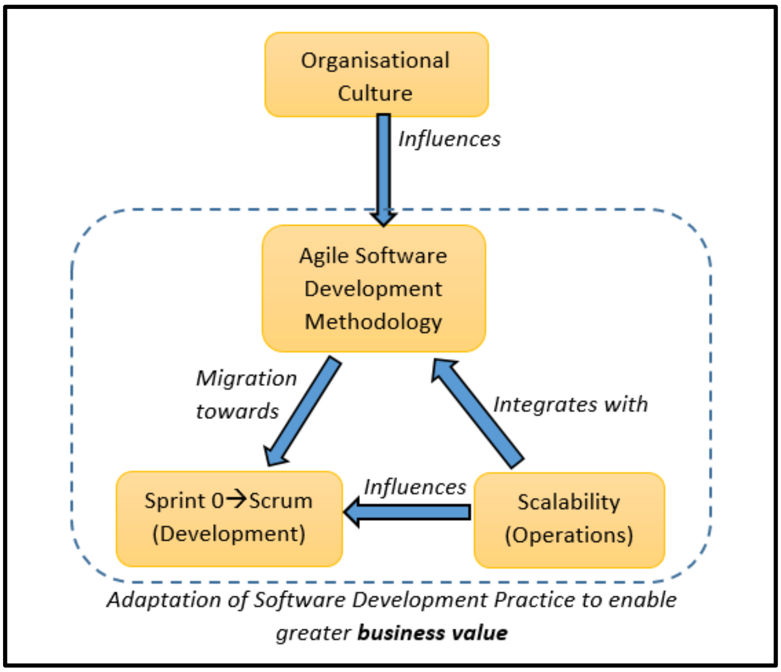

Organisational Culture
Influences
Agile Software Development Methodology
Migration towards
Integrates with
Sprint 0→Scrum (Development)
Influences
Scalability (Operations)
Adaptation of Software Development Practice to enable greater **business value**

Figure 5.12: The Study's *Trinity*

The relational format adopted in Figure 5.12 is used to display the researcher's interpretive effort to 'join the dots' and present a cogent overview of the main constructs that have emerged from the empirical evidence presented thus far. The influence of organisational culture on the methodology used for software development in an organisation has been accorded the highest priority because it has a cascading influence on all SPI initiatives. The organisational imperative to ensure that the software development process is aligned to the generation of business value has influenced South African organisations to adopt an agile approach to software development. The preferred agile methodology of choice in South Africa is Scrum, although there are many variants of the methodology that are currently being used. The customisation of Scrum has been achieved by combining Scrum methods with Waterfall-like methods as well as XP and Kanban methods so that the development team is placed in a 'comfort zone' that enhances optimum productivity. The proviso for Scrum based development is that it is preceded by an upfront planning session that is undertaken with varying levels of intensity that depends on the type of project as well as the culture in an organisation. The biggest impediment to Scrum as a methodology is its inability to scale up to the operational environment at infrastructure/organisational level. In order to address the issue of scalability, many South African organisations have resorted to a DevOps initiative where the intention is to reduce or eliminate the 'disconnect' between development and operations. This initiative requires greater collaboration between all stakeholders involved in the software process. Under this new collaborative arrangement, software development teams are required to abandon the 'sanctity of the Scrum development space' and change development habits to embrace stakeholders such as business analysts, operations and security engineers and quality assurance personnel during development cycles. The adoption of a collaborative environment for software development requires the invocation of a software lifecycle approach that enhances collaboration and ease of access to the various stakeholders involved in the development process. The complexities of enhancing the value obtained from such a socio-technical environment is best understood by firstly 'unpacking' the oversight influence that organisational culture has on the adoption of a software development methodology in an organisation.

5.5.1 The Influence of Organisational Culture (OC) on the level of Agility

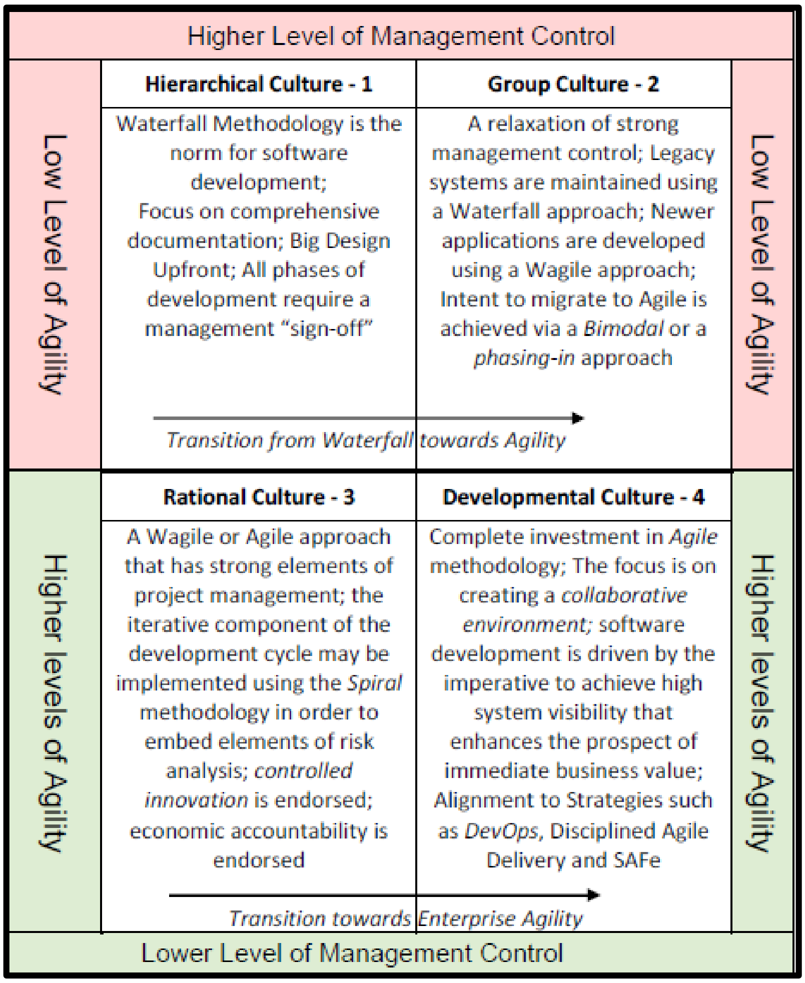
The strategy adopted in the current study to operationalise the abstraction inherent in OC is to make use of the Competing Values Framework (CVF) that was introduced in Section 2.5.2 of Chapter 2. An overview of the CVF is presented for quick reference to enable comprehension of the discourse on OC and the level of agility that may be adopted in an organisation.

A Recap of the Competing Values Framework (CVF)

Quinn and McGrath (1985) proposed the CVF as a theoretical model to operationalise the amorphous concept of OC. The CVF has been adapted by Denison and Spreitzer (1991) to classify OC according to one of the following 4 dimensions. A *Hierarchical Culture* where there is a focus on stability and management 'command and control', a *Group Culture* where the focus is on control and monitoring of employee alignment to a set of prescribed processes derived from historical organisational practice, a *Rational Culture* that has a focus on innovation, productivity and accountability with regards to resource consumption and a *Developmental Culture* that has a focus on innovation and the generation of new ideas.

Based on the interpretation of the empirical data with regards to OC, a framework (illustrated in Figure 5.13) was developed to guide the adoption of a software development methodology classified according to the prevalent culture in an organisation. The model presented in Figure 5.13 provides a 'roadmap of change' from a purely Waterfall-driven approach to an agile-like approach that may eventually culminate in a full-blown agile approach. The cultural shift that is a prerequisite for an agile approach entails a migration from a cultural disposition that prioritises predictability and order to one that embraces spontaneity and flexibility as enablers of competitive advantage. These 2 competing paradigms of cultural philosophy is further classified according to 4 cultural types (from

Denison and Spreitzer (1991)) that have each been matched to a methodology of software development (an output from the empirical analysis conducted in the current study) and illustrated in Figure 5.13.



Higher Level of Management Control

Low Level of Agility

Hierarchical Culture - 1

Waterfall Methodology is the norm for software development; Focus on comprehensive documentation; Big Design Upfront; All phases of development require a management "sign-off"

Group Culture - 2

A relaxation of strong management control; Legacy systems are maintained using a Waterfall approach; Newer applications are developed using a Wagile approach; Intent to migrate to Agile is achieved via a *Bimodal* or a *phasing-in* approach

Transition from Waterfall towards Agility

Higher levels of Agility

Rational Culture - 3

A Wagile or Agile approach that has strong elements of project management; the iterative component of the development cycle may be implemented using the *Spiral* methodology in order to embed elements of risk analysis; *controlled innovation* is endorsed; economic accountability is endorsed

Developmental Culture - 4

Complete investment in *Agile* methodology; The focus is on creating a *collaborative environment;* software development is driven by the imperative to achieve high system visibility that enhances the prospect of immediate business value; Alignment to Strategies such as *DevOps*, Disciplined Agile Delivery and SAFe

Transition towards Enterprise Agility

Lower Level of Management Control

Figure 5.13: Level of Agility Classified according to Organisational Culture

The OC classifications of the CVF have been reconfigured and presented in Figure 5.13 according to the parameters of *management control* and the *level of agility*. In Quadrants 1 and 2 (upper level quadrants in Figure 5.13), there is a high level of management control. The culture classification in Quadrant 1 is the Hierarchical Culture which is the antithesis of agile methodology. The *Waterfall methodology* for software development resonates quite well with an organisational culture that is hierarchical (also confirmed in the frequency cross analysis from

Figure 5.12) and places high value on predictability and order. A discernable trend in South African organisations is the shift away from a hierarchical culture where a 'command and control' style of management is being phased out in preference for a more democratised approach that embraces controlled innovation and a 'relaxing' of management control. The compromised stance towards management control is seen as an attempt to yield positive results such as higher staff morale and employee satisfaction. The following verbatim excerpt attests to this claim.

> *At first I would say that it was quite hierarchical and the methodology of choice was Waterfall. However, recently with the hype around agile methodology and the training sessions on the use of Scrum the management influence is not that great and developers have a greater freedom to express themselves.* (***Interview 6, Software Engineer in the Banking Sector)***

The Group Culture (in Quadrant 2 of Figure 5.13) also espouses a high level of management control. However, this culture is a lot more dynamic and there is a prominence of change management structures that enable controlled changes to be made to existing procedures and protocols. The change management structures arguably ensure that change is gradual, highly controlled and subjected to management scrutiny to ascertain the viability of implementing the change. The use of terms such as *Wagile* and *Bimodal* are symptoms of the prevalence of a Group Culture. The Wagile approach, which is a compromise between a plan-driven approach and an agile approach, incorporates a high level of upfront planning followed by agile methodology such as *Scrum* for the actual coding phase. In a South African context, the Group Culture is one that employees feel quite comfortable with because at various stages of the organsational history, employees make innovative contributions that are subjected to organisation-wide scrutiny. Once the innovation is implemented, employees are responsible for 'championing' the innovation until it becomes successful and fully entrenched as part of the organisational set of procedures or as part of organsational behaviour. The preceding assertion is corroborated by 2 verbatim excerpts attesting to a preference for a Group oriented culture in the context of agile and DevOps adoption respectively.

> *I also think Group Culture is unavoidable because you cannot expect an organisation to be fully Developmental over a sustained period of time because this will cause a chaotic situation. So if you have adopted a Developmental Culture so that Agile can work well, then after a period of time, this becomes a new culture and that you may now say is Group Culture …so when you have new staff members, they will have to adapt to the norm which is now a Group Culture.* ***(Interview 8, Senior Business Architect)***
>
> *…remember if it is a new way of doing things, it inevitably ends up a culture within an organization. So if it is a DevOps culture that we are trying to build, agile culture that we trying to build, if you can't fit in, you don't belong***. (Interview 14, DevOps Manager in the Banking Sector)***

The Group Culture still maintains a high level of management control where the focus is on ensuring that *established* organisational processes are followed quite rigidly. Hence, it cannot be accorded a status of being fully compatible with agile methodology because the Group Culture can easily promote a highly prescriptive environment that could degenerate into a Hierarchical Culture. It does however, represent a form of OC that enables a transition from Waterfall methodology to a 'diluted' version of Agile Methodology. This theory is aligned to the suggestion by Denison and Spreitzer (1991) who claim that a change in organisational processes can only be implemented successfully if the change agent is able to understand and transform the underlying values and assumptions that underpin organsational behaviour. In the context of software process improvement initiatives, there has to be greater value accorded to the innovative suggestions made by an organisation's software development practitioners. Once this imperative is achieved, the culture within the organisation gravitates away from a Hierarchical orientation thereby creating a path for the adoption of agility

The Rational Culture orientation is less prescriptive than the Group Culture orientation and highly compatible with agile values and encourages innovation and responsiveness to change from a software development methodology perspective. Software developers are not micro-managed in terms of adherence to a specific methodological approach. Development teams are encouraged to be innovative and customise a development methodology according to the requirements of the project. There are variant strains of agility in terms of development strategy. As an example, the intensity of upfront planning and design is left to the discretion of the development team. However, attached to the flexible environment is the requirement of accountability with regards to resource consumption. While project management is not necessarily invoked via traditional project management protocol, the development teams are expected to be self-managing not only in terms of progress with the development effort but also in terms of time and resource consumption. The Rational Culture is also driven by an imperative to ensure that the business value inherent in the systems development project is not compromised. Hence a *Spiral* approach is ideal because after each release of a version of a system, a cost-benefit risk driven analysis is undertaken to facilitate the attainment of business value is still on track and resource consumption is within the expected parameters.

The final quadrant in Figure 5.13 is reserved for Developmental Culture which represents the highest form of agility. The Developmental Culture is strongly aligned to the Scaled Agile Framework (SAFe) discussed in Section 2.5. The SAFe requires a complete shift in organisational thinking that requires an enterprise-wide effort to endorse an agile approach not only in the context of software development, but also general business decision making and problem solving. Under this environment, where an organisation reaches full agile maturity, the true benefits of a DevOps approach will be achieved. In a South African context, based on the empirical evidence provided by virtue of the interview data, none of the practitioners have indicated that their organisation had achieved this highest form of agile maturity. A distinguishing feature of Developmental Culture is that management control is low and the level of agility that may be achieved is high. In order to attain this level of agile maturity there has to be a high level of trust that is bestowed upon employees. A verbatim comment attesting to this phenomenon is provided below.

> *If agile methodology ensures that quality systems are built and all requirements are met and the system is delivered on time and within*

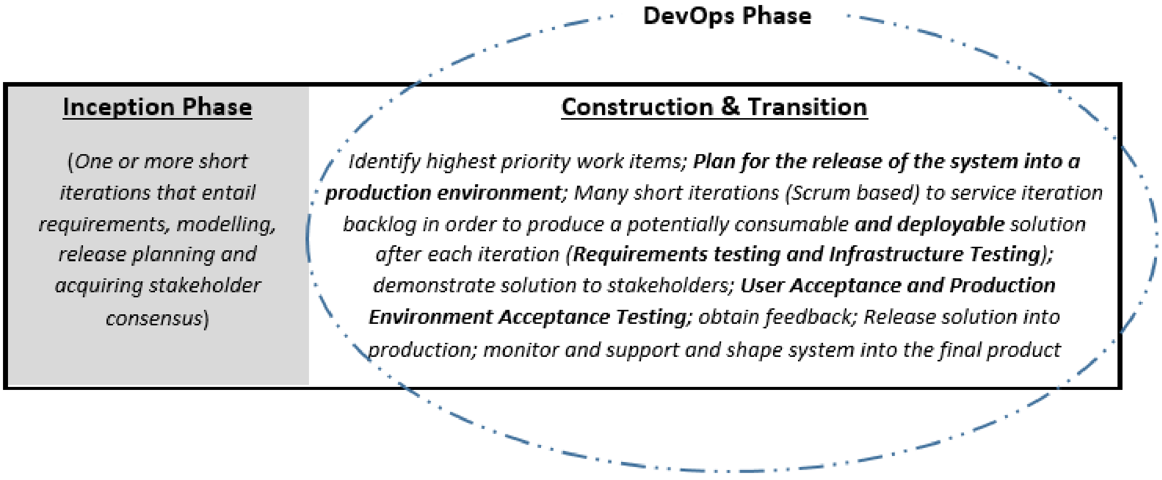
> *the allocated budget …and if that happens regularly then the perception of an organisation being hierarchical will not be that great and more trust will be conveyed to the IT people… in which case a developmental and rational culture may become the order of the day.*
> ***(Interview 3, Business/Systems Analyst in banking sector)***

5.5.2 Addressing the Technical Dimension of Software Development Methodology

Based on the empirical evidence provided, the two major technical areas that were identified are:

- the endorsement of Scrum as the default methodology of software development with the proviso that there is an appropriate intensity of effort accorded to upfront planning;
- the need to integrate operations processes into the development methodology to enable a smooth transition of the system from a development environment onto the production environment thereby enabling quicker realisation of business value.

A possible 'ready-made' solution that meets these requirements from a methodological perspective is the Disciplined Agile Delivery (DAD) model proposed in Ambler and Lines (2012, p. 12) and presented and critiqued in the current thesis in sections 2.4.3 and 2.4.4 respectively. From an overview perspective DAD differentiates the software process into 3 phases. These are the inception, development and transition phases. The *inception* phase consists of the upfront planning which may be further decomposed into business and the software specifications. The *development* phase consists of Scrum based methodology to handle the coding and testing of the evolving systems and *transition* is a reference to the deployment of the system onto a production environment. The inception phase is given upfront priority and is handled by the business analysts, the systems analyst and the product owner to create a set of development processes and protocols that are 'intuitively shaped' to align with the culture of the organisation. The output of the inception phase is a plan that guides the development and transition phases. Based on the empirical evidence from the current study, the inception phase has been identified to play a pivotal role in systems development and there has been a substantial effort made by organisations to derive a process model that works in the context of that organisation. From a South African context, this phase is driven by the Business Analysts (BA's) (4 interviewees have served in this capacity) and has become widely acknowledged as a phase where the business requirements are specified to the software development team, usually represented by the Product Owner (PO) (in an agile context). The liaison between the BA's and PO's has become well entrenched and is not viewed as a problematic phase of the development process. The arrangement between *development* and *transition* has been identified as an area of concern because of the inability of Scrum to enable the attainment of the expected business value within the expected timeframe. In order to address this situation, an adapted version of DAD is used to provide an overview of the proposed solution. The illustration in Figure 5.14 is based on the DAD model. The main area of modification is the conflation of the *development* and *transition* phases into a single DevOps phase, illustrated in Figure 5.14.



DevOps Phase

Inception Phase

(One or more short iterations that entail requirements, modelling, release planning and acquiring stakeholder consensus)

Construction & Transition

*Identify highest priority work items; **Plan for the release of the system into a production environment**; Many short iterations (Scrum based) to service iteration backlog in order to produce a potentially consumable **and deployable** solution after each iteration (**Requirements testing and Infrastructure Testing)**; demonstrate solution to stakeholders; **User Acceptance and Production Environment Acceptance Testing**; obtain feedback; Release solution into production; monitor and support and shape system into the final product*

Figure 5.14: Adaptation of the DAD Model

Based on the empirical evidence, the organisations have achieved moderate success in the transition from a Waterfall methodology to Agile Methodology by focusing efforts on reducing the divide between business and software development through the establishment of roles such as the PO and BA. Although the 'silo mentality' still exists in many organisations, the PO and BA are pivotal in ensuring that the business imperative of the software development process is upheld. The PO's and BA's enable collaboration between the 'business silo' and the development and quality assurance (QA) silos as illustrated in Figure 5.15.

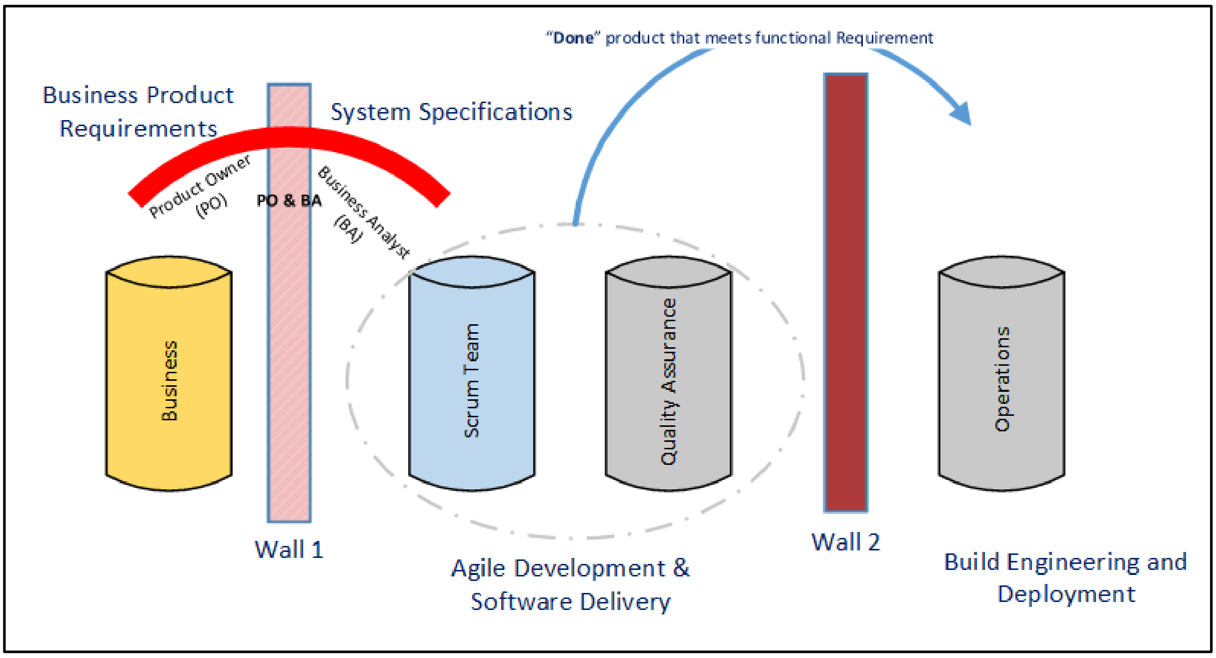

"Done" product that meets functional Requirement
Business Product Requirements
System Specifications
Product Owner (PO)
PO & BA
Business Analyst (BA)
Business
Scrum Team
Quality Assurance
Operations
Wall 1
Agile Development & Software Delivery
Wall 2
Build Engineering and Deployment

Figure 5.15: The Silo Based Approach to Agile Software Development

This divide between the various stakeholders in the software development process is illustrated in Figure 5.14. Business is logically separate from development although the PO's and BA's manage to ensure that the Wall 1 divide is circumvented by conveying business requirements to the development team in the form of the system specifications document that contributes towards the Product Backlog. There is also a close working relationship between the development teams and QA. In many instances the Scrum development team members are responsible for running unit tests thereby conflating the roles of developer and tester. The formal reviews and inspections are conducted by PO's, BA's, developers and testers who collectively perform the QA function.

The glaring omission from this collaborative initiative is the lack of clarity on the role of the operations staff, the people who are responsible for the technical alignment of the newly developed application into the 'organisational ecosystem' or the organisational infrastructure. Once a new system is commissioned by the business division, the functional requirements are conveyed to the development teams by the PO's and BA's. The development teams develop the system iteratively and add functionality in an incremental manner using Scrum oriented methods. As each increment is endorsed by QA, the expression 'done' is used to indicate the completion of a task. Once all increments have been completed and the required functionality is achieved to the satisfaction of QA, the newly developed application is 'tossed over the wall' to the operations people who are responsible for the build engineering phase of the application's development lifecycle. The build engineering phase is traditionally regarded as a separate 'silo' in the process of getting the application into a 'live'/ production environment. The empirical evidence gathered thus far seems to converge to the perspective that there is not much support for the operations phase and once the application traverses Wall 2 (illustrated in Figure 5.15), a 'bottleneck' situation is created because of the lack of resources available to the operations staff. However, the bigger impediment to productivity is the lack of collaboration between the development team and the operations team when it comes to tackling system integration problems. Once a system is handed over to the operations staff, the Scrum team is disbanded and allocated to other projects. The operations staff are then saddled with the task of fixing operational errors typically linked to the network and security infrastructure of the organisation. The operational error fixes are compounded by the sparse documentation that is generated as well as the non-availability of members of the development team. In order to address the impasse between development and operations, an incursion into realms of software build engineering is necessitated.

The Build Engineering Phase

The discourse on Build Engineering is based on interviews that the researcher has conducted with the following experts in the field of software operations and DevOps. These interviews were not part of the main data corpus. However, the initial data analysis converged to an outcome that necessitated an engagement with operations engineers. In all 3 instances, the experts who were interviewed agreed to have their identities revealed in the study's report.

- Bob Aiello – American based chair of the IEEE Working Group on DevOps and co-author of the book Agile Application Lifecycle Management (*referenced as* Aiello and Sachs (2016)) and expert in the domain of software engineering and DevOps (*the interview* is *referenced as* Aiello (2017));
- Brad Black – American based Scrum Coach and expert in the domain of software engineering and agile software development (*referenced as Black (2017)*);
- Jonathan Frankel - An experienced DevOps Engineer at a leading bank in South Africa where there has been a major process re-engineering effort to alleviate the bottleneck situation that prevails at operations level by adopting a DevOps strategy (*referenced as* Frankel (2017)).

According to Aiello (2017) software development has to have a complete lifecycle approach and it is misleading to speak about a software development methodology that functions in isolation of the context in which the software is developed. The context from an organisational perspective includes BA's, QA, project managers, security managers, developers, testers, operations engineers and end users. In order to achieve this all-inclusive environment, a possible strategy is to pack each sprint with functional requirements as well as a continuous integration (CI) and continuous delivery (CD) imperative. This viewpoint is supported by Black (2017) who is of the opinion that currently, software development occurs in an environment where there are many "moving parts" and in order to account for the various influences on the development process, an agile scrum based approach is required together with a DevOps culture to enhance the prospect of the business value planned for a software system is achieved and delivered in a short space of time. The main benefit of adopting a DevOps approach is that it will enable the breaking down of the traditional silos that impedes software development productivity and enables a lifecycle approach that extends from inception to release of the system on a production server. These sentiments also concur with the perspective of Frankel (2017) who endorses a DevOps approach to software development because it enables a collaborative environment that is structured so that all stakeholders in the software development process are easily accessible thereby ensuring ease of access to cross-functional knowledge. As a best case scenario, Frankel suggests that all the stakeholders should be co-located especially where there is a critical time based constraint to deliver a system. However, he does concede that this may not be practically feasible and a more dynamic approach that enables quick access to all stakeholders at instances when they are required will be ideal.

At this juncture the researcher realised the need to make an incursion into the domain of Build Engineering in order to comprehend the full essence of operations work.

According to Aiello and Sachs (2016, p. 91) Build Engineering is the discipline of efficiently converting source code into binary executables that is in a state of readiness for deployment to the underlying technology infrastructure. This is normally achieved by running scripts that are created using technologies such as Ant, Maven or Make so that the process is repeatable and quick. These technologies were also mentioned by Frankel (2017). There is currently a trend for the development team to run these scripts that are then deployed to a test environment, and not the actual production/live environment. The Build Engineer (BE) performs the operations task of deploying the application to the production environment. However, as Aiello and Sachs (p. 93) point out, this process can become quite complicated and the Build Engineer is required to make an intervention on the development side to rectify incompatibilities or bugs that were not identified during QA. Quite often this entails an incursion into development technologies such as the .Net, Java or COBOL platforms. The complexity of the task is exacerbated by poorly written source code or when the build activity is undertaken as a ‘big bang’ deployment rather than an incremental one (Aiello, 2017). Aiello and Sachs (2016, p. 97) make the salient point that the Build Engineers need to engage with the developers early in the development lifecycle so that there is convenient availability of “deep knowledge” of the system from a developmental and technical perspective. Organisations that adopt a DevOps strategy tend to curate an environment that enables easy collaboration between the various software development stakeholders. In reference to the OC classification provided in Figure 5.13, the DevOps strategy resonates well with a cultural environment that is classified as Developmental.

The path taken by the current study started with a focus on the inner workings of software developmental methodology and the perspectives provided in this regard by experienced software engineers. The outcome of the analysis of empirical evidence provided thus far has resulted in the study's gravitation towards the operations domain. In the annals of software engineering academic literature there is sparse coverage of operations methodology and minimal reference to the 'tool stack' used by operations engineers. The interviews conducted with Frankel (2017) as well as Aiello (2017) coupled with information from Aiello and Sachs (2016) is used to mitigate this situation. The empirical evidence gathered from these sources are suggestive of a systems lifecycle (illustrated in Figure 5.16) that consists of development, continuous integration (CI), continuous delivery (CD) and deployment (and monitoring). The illustration in Figure 5.16 to represent the development, integration, delivery and deployment pipeline is based on ideas pioneered by Humble and Farley (2010).

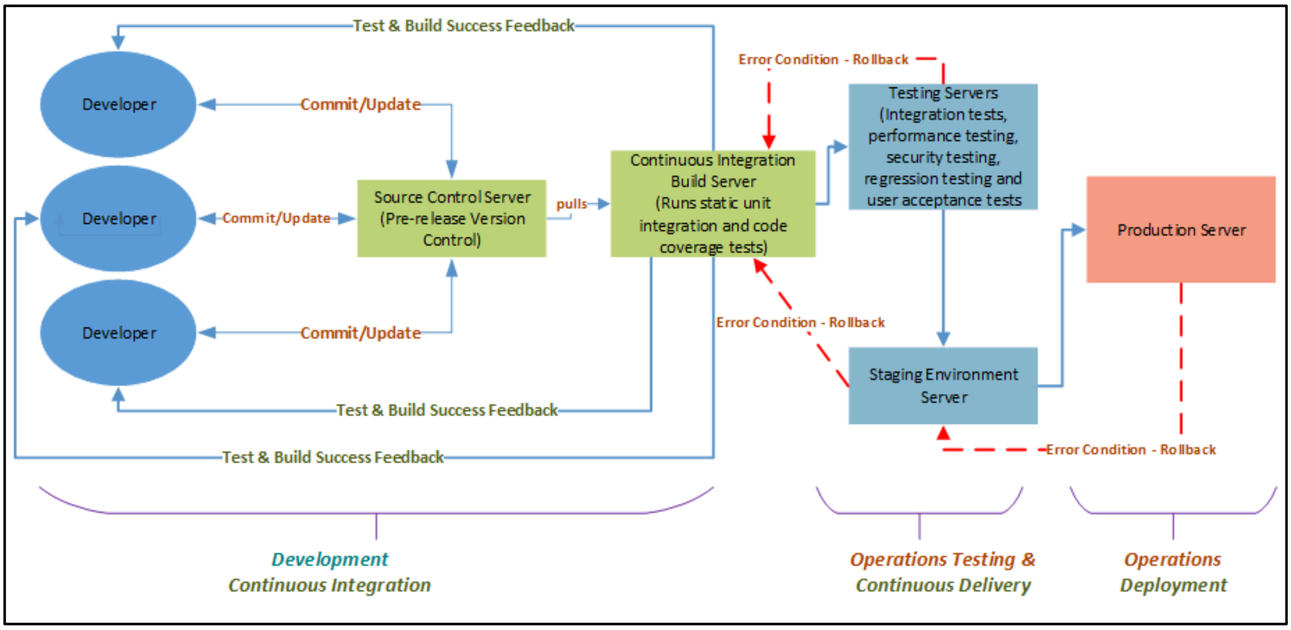

Test & Build Success Feedback
Developer
Developer
Developer
Commit/Update
Commit/Update
Commit/Update
Source Control Server (Pre-release Version Control)
pulls
Continuous Integration Build Server (Runs static unit integration and code coverage tests)
Error Condition - Rollback
Testing Servers (Integration tests, performance testing, security testing, regression testing and user acceptance tests
Production Server
Error Condition - Rollback
Staging Environment Server
Error Condition - Rollback
Test & Build Success Feedback
Test & Build Success Feedback
Development
Continuous Integration
Operations Testing &
Continuous Delivery
Operations
Deployment

Figure 5.16: The Development-Deployment Pipeline

The phases of the Development-Deployment pipeline are explained below.

- The first phase is the actual development undertaken by the software developers. The main deliverable from this phase is the contribution of incremental functionality towards the development of a system. The evolving system is referred to as the baseline system and developers are required to contribute towards the incremental enhancement of the baseline system by adding functionality that is the output from sprint cycles. According to Frankel (2017) developers are encouraged to 'commit' code towards the baseline as often as possible. This activity of committing code towards the baseline is referred to as **continuous integration** (CI). The management and co-ordination of the 'commits' towards the baseline system is done via automation servers and tools. The tools that are popularly used currently for this process in South Africa are Git (an open source software version control tool) and Team Foundation Server popularly used on Microsoft platforms. A significant observation with regards to software development teams in South Africa is that the practice of *code commits* to the baseline system has only been recently identified as an area of urgency. Previously the code commits were quite infrequent and the developers did not have a compulsion to commit their code to a central repository on a regular basis. The current imperative is to commit code as often as possible during the course of each day and at least once a day. However, Aiello and Sachs (2016) do warn against too many builds in a short space of time because it creates a destabilising environment for the developers. The code baseline is constantly monitored by the CI server for any changes to the baseline code. If a change is detected, the code merge tests are immediately executed and feedback is provided to the developers. This practice arguably ensures that 'code merge' defects are detected early which can then be handled on a continual basis rather than a 'big bang' approach where all defects are handled at the end of the sprint, at which point the system has acquired "…a level of complexity that is not easily tamed" (Aiello, 2017). The testing environment that detects build related issues linked to merge conflicts or possibly conflicts with the different environments in which the application will be installed is referred to as the CI server and the most frequently mentioned server by all 3 interviewees is the java based open source server named Jenkins. All errors identified by the CI server are sent via an error report to the members of the development team;

- The 2nd phase entails the delivery of the systems that have undergone a successful build and has been released for QA testing as well as staging tests. This is referred to as the **continuous delivery** phase were the code baseline is in a state of readiness to be deployed to the production server at any time;
- The final phase entails deployment of the system onto the organisation's production servers. This phase is referred to as the **Deployment** phase and is usually not a continuous process. The immediate release of newly developed systems via automated processes may be impractical for pragmatic reasons. The main reason is the requirement to release new features to the end user base in a controlled manner thereby ensuring that end users are not overwhelmed with changes to the system in short time intervals. Another reason could be the culture orientation within an organisation. This may be classified as Hierarchical or a Group Culture where there has to be extensive 'sign offs' before a system feature is implemented on a live production environment. Based on the empirical evidence gathered, the *signing off* activity is a major component of the systems development lifecycle. In organisations that are transitioning to an agile methodology, the signing off phase is quite dynamic because business managers who are entrusted with

this responsibility have a close working relationship with the development teams.

As suggested by Frankel (2017) and Aiello (2017) the recent trend which is aligned to the adoption of a DevOps approach is to try and achieve maximum automation in the development, build and deployment pipeline. This was not the case previously when the strategies of continuous integration and continuous deployment were not practiced.

The integration of software deployment requirements into the development lifecycle (a reference to a 'left shift') is an idea that has been endorsed by both Jonathan Frankel and Bob Aiello. Whilst the current strategy is to aim for maximum automation via the development-deployment pipeline illustrated in Figure 5.16, the entire framework still has a 'silo-oriented' appearance that is essentially sequential in nature. This situation becomes quite apparent when the process is reviewed via a Sequence Diagram as is illustrated in Figure 5.17 where the need to have a *left shift* from the 'Ops' perspective becomes quite apparent. The 'internal' development, testing and integration seems to be well handled via the sequence illustrated in Figure 5.17. However, the delivery of value to the end user/customer is only made possible after the involvement of the operations staff.

A significant outcome of the researcher's conversation with Aiello (2017) and Black (2017) is that the consequence of not following a continuous integration plan that includes operations staff as part of the software development is that the expected benefits of following an agile approach to software development is not achieved in an optimal manner. This assertion is corroborated in the South African context by Frankel (2017) who makes reference to initiatives within his organisation to try and address this issue.

The first principle based on the Agile Manifesto is an attachment of the highest priority level to the imperative to "*...satisfy the customer through early and continuous delivery of valuable software*" (Fowler & Highsmith, 2001). However, the focus on continuous delivery seems to have a blurred interpretation within the confines of software development nomenclature. This assertion is based on the empirical evidence which suggests that the agile-based term *done* alludes to the activity of development and internal testing *within* the software development environment and not the actual deployment environment. This situation is illustrated in Figure 5.16 as the internal development cycle (illustrated as code commit cycle/loop) and Figure 5.17 (cycle to the end of the Sprint phase and the achievement of 'done'), where the activity of development is focused more on the software system and the pre-defined requirements tests.

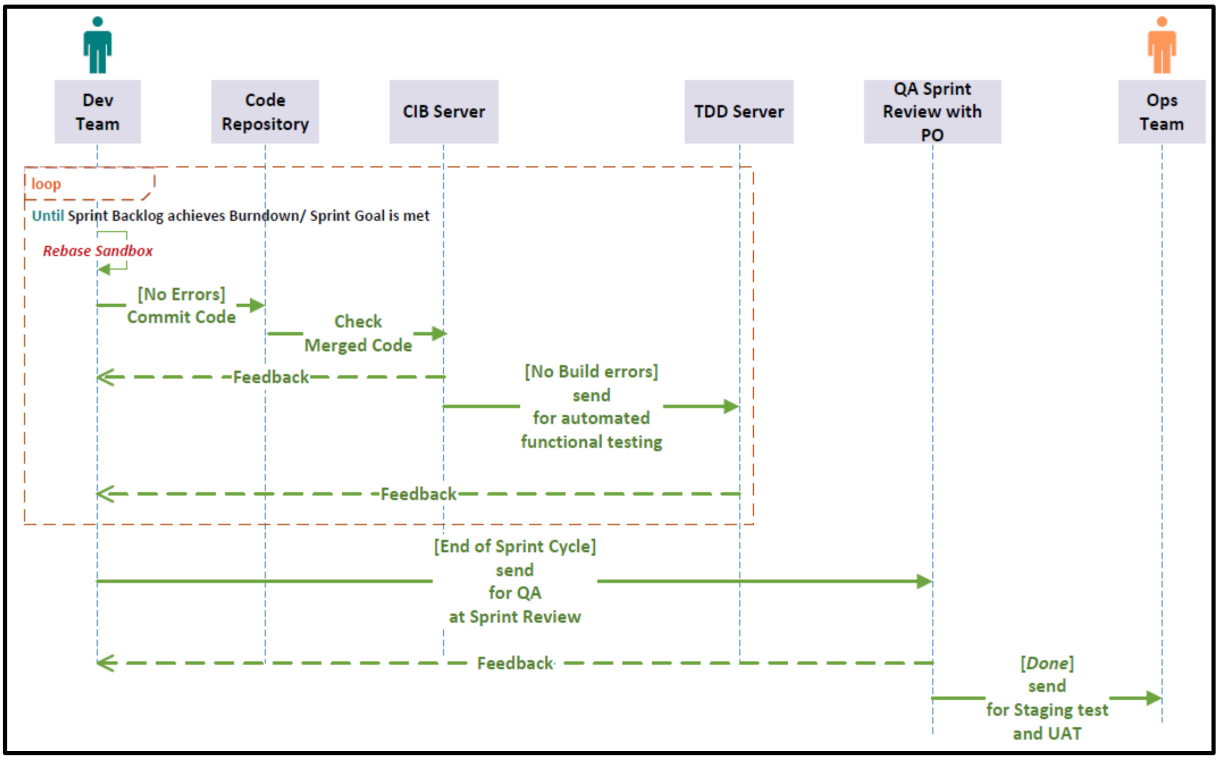

Dev Team
Code Repository
CIB Server
TDD Server
QA Sprint Review with PO
Ops Team
loop
Until Sprint Backlog achieves Burndown/ Sprint Goal is met
Rebase Sandbox
[No Errors] Commit Code
Check Merged Code
Feedback
[No Build errors] send for automated functional testing
Feedback
[End of Sprint Cycle] send for QA at Sprint Review
Feedback
[Done] send for Staging test and UAT

Figure 5.17: The "Internal" Development, Integration & Testing Cycle

The system is not viewed as a product or a 'business commodity' that has a strong organisational context where it contributes to the attainment of business value. Based on the evidence from the initial set of interviews, 86% of the interviewees made mention of the requirement for the software development process to deliver business value. However, this 'business value' is not being delivered with the immediacy envisioned by the transition to agile methodology. At this juncture the empirical phase of the study has been pivotal in establishing a possible “Achilles’ heel” in the agile software development process. This is the lack of continuity between the development and the deployment phases of the software process.

The ‘disconnect’ between software development and the attainment of value from a software product has been addressed by invoking techniques such as Continuous Integration (CI) as illustrated in figures 5.16 and 5.17. The practice of CI is endorsed as a core activity of agile software development and is documented by the Agile Alliance as an initiative to ensure that software that is tagged as *done* should be available for immediate release into a production environment. CI is also an integral method of the XP oriented approach to software development. However, the complexities of implementing CI and achieving a state whereby software produced from the agile development phase may be tagged as ‘immediately releasable’ are not easily attained as explained in an article by Martin Fowler (see Fowler, 2006), one of the authors of the Agile Manifesto. Fowler suggests that the problem lies with the lack of focus on the activity of CI thereby making continuous delivery and deployment difficult to achieve. This problem is given extensive coverage in Aiello and Sachs (2016) and Humble and Farley (2010).

The problem of the lack of continuity between development and deployment has been the subject of interventions that entail the use of tool-based support for CI. The current availability of sophisticated tools from the open source community (such as Git and GitHub) and vendor based tools (such as Team Foundation Server) has made the strategy of CI a feasible option because of the minimal overhead that is incurred to arguably ensure that developers engage in the practice of updating the code base on a regular basis. The problem is two-fold.

1. CI is restricted to the development environment which is not an accurate reflection of the production/live environment. CI ensures that the functional requirements are met and then the system is handed over for delivery and deployment to an environment that is not identical to the development environment.

2. CI is not a formal part of the Scrum 'ceremony' and it needs to be incorporated as a Scrum method. This will ensure alignment with the agile principles that espouse CI and Continuous Delivery.

The Development and Deployment Dilemma

According to Aiello (2017) and Frankel (2017) it is not easy to fully align the development environment to the production environment. The reasons for this dilemma are elaborated in Aiello and Sachs (2016) as well as Humble and Farley (2010, p. 105). The 'development-deployment' dilemma was previously identified by Jez Humble[22] as a possible aspect of the development lifecycle that could impede the prospect of agile software development methodology from upholding those principles that relate to the frequent delivery of working software to its end user base.

However, as Humble (2017) points out, the topic of software deployment has not received much attention from academic circles although it has become a major source of concern in industry where it is still not very well understood. Currently, the practitioner community is making a concerted effort to address this situation by ensuring that developers at least engage in the practice of CI. According to Frankel (2017) his organisation has requested for development teams to ensure that they 'commit' code to the baseline system as often as possible and to maintain the baseline code in a state of stability. McConnell (1996, p. 144) used the expression "maintaining the heartbeat" of the system to describe CI efforts that ensure that the baseline system is in a stable state. Both Fowler (2006) and Humble and Farley (2010) provide clear directives on how to ensure that the baseline system is maintained in a state of stability. The process entails daily code commits from the developers coupled with an effort to ensure that the code baseline is maintained in a stable state. If the stability of the code baseline is compromised, then this should be flagged as a problematic situation and all further development has to stop. The alternatives at this juncture is to try and achieve a quick fix to the problem or to roll back the code baseline to a stable state.

[22] Jez Humble received the 2011 Jolt Award for his contributions to software engineering excellence and his co-authorship of the book titled Continuous Delivery

As can be ascertained from the preceding paragraph, CI entails a substantial effort on the part of the software development team. However, Scrum methodology does not make direct reference to any form of CI as an integral part of the methodology. The pivotal role played by CI cannot be ignored by the methodological component of Scrum software development practice. The incorporation of CI practice into the Scrum ceremony may be seen as part of the evolutionary trajectory of agile software development methodology where software process improvement techniques have to incorporate aspects from the delivery and deployment phase. Aligned to the afore-mentioned imperative to adjust software process models so that CI is incorporated as part of the development process, the de facto model representing the Scrum-oriented software process has been modified to incorporate activities that enhance a DevOps approach to software development.

According to Aiello (2017) the activity of Build Engineering should form an integral part of the agile software development process. Aiello and Sachs (2016, p. 91) define Build Engineering as the process of converting source code into binary executables that may then be run on any platform. The need for Build Engineering is linked to software developer habits which entail a preference for using an integrated development environment (IDE) and a software development platform with which they have acquired substantial familiarity. This assertion is verified empirically by the study's core data collection phase where it has been established that developers make use of varying technologies, frameworks, platforms, IDEs and programming languages. There are instances where team members make use of different IDE's whilst working as part of a single team. Hence, the heterogeneity of the development platform becomes a source of challenge for the Build Engineer (BE) who has the task of identifying the compile and runtime dependencies of the development environment so that an appropriate binary executable may be created for the test environment. Aiello and Sachs (2016, p. 98) provide a compelling imperative for the need to include the topic of Build Engineering in any discourse on software process improvement. The argument made is that developers usually engage in the development, build and test cycles without any consideration for the actual deployment environment. The ideal situation is to have the build and test environments that are as 'identical as possible' and as 'close to production as possible'.

In order to mitigate this problem Aiello (2017) makes the suggestion that "…as a Build Engineer, I would like to sit with the developers so that I can learn how the system works and I am plugged into the flow of the development effort." This suggestion is also endorsed by Frankel (2017) who intimated that a similar approach was followed in his organisation as part of the DevOps strategy. The point being made is that Build Engineering planning should be incorporated early and continually in the development lifecycle. From an agile perspective, this could occur at the Sprint planning phase where the BE (serving in the capacity as a representative from the Operations team) is provided with details of the development platform so that the testing and production environment could be configured to enable compatibility between development and operations. A proposed solution that integrates Build Engineering into the Scrum development cycle is illustrated using a cross-functional flowchart model showed in Figure 5.18.

The underlying theory behind the model illustrated in Figure 5.18 (referred to as the Scrum Development Operations Model (SDOM)) is that the invocation of a DevOps approach is only possible if there is a 'left shift' (Aiello & Sachs, 2016, p. 223) of the operations function into the development domain. As illustrated, the 'Product and Sprint Backlog' swim lane is identical to the original Scrum model (presented in Section 2.4.5 of Chapter 2). However, adjustments to the original Scrum model is based on the practice of continuous integration which have been added to the traditional processes associated with Scrum development. Also, an additional layer has been added to mitigate the risks attached to a pre-mature product increment release that has not been completely tested in a 'production-like' environment.

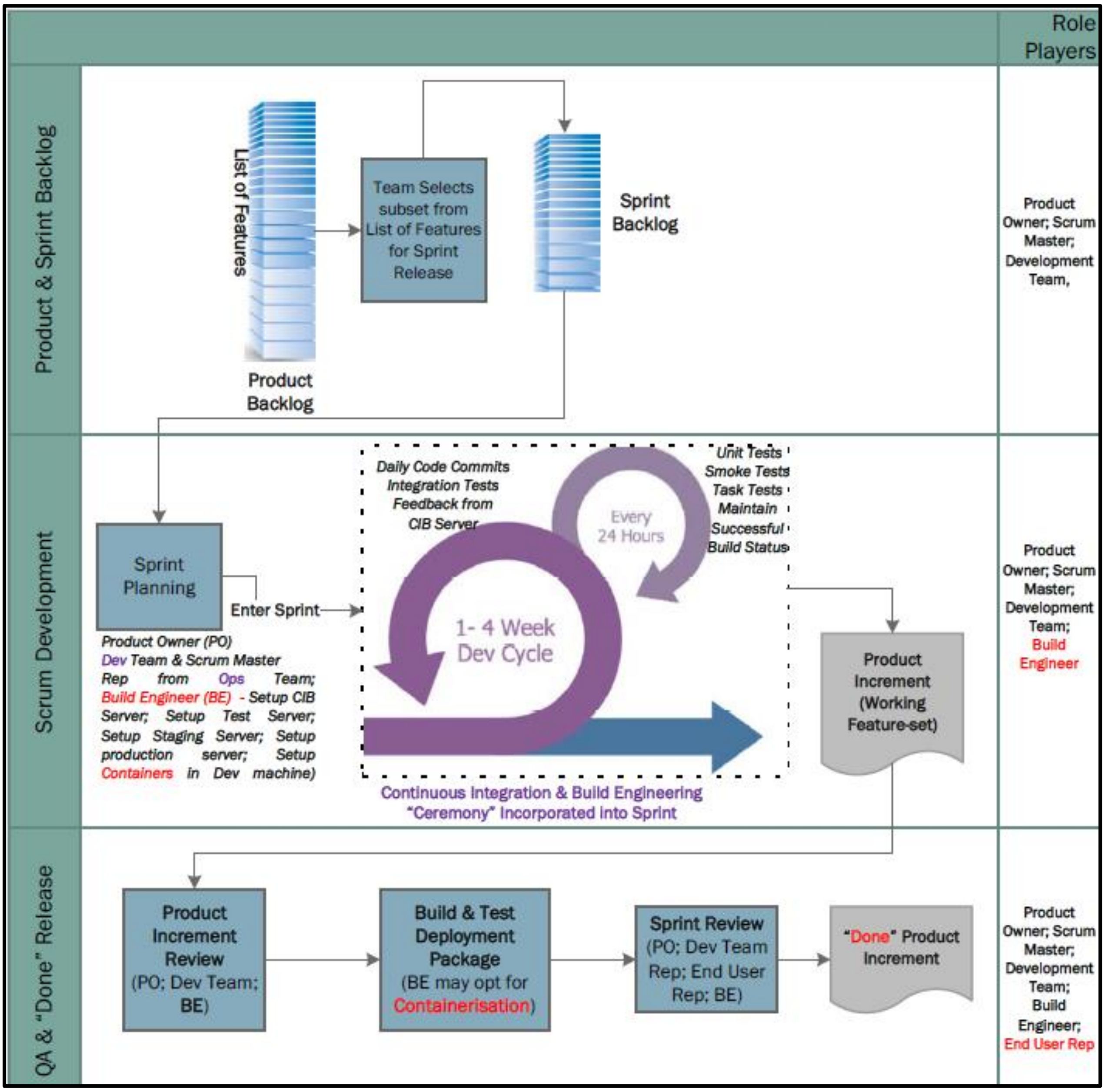

Role Players
Product & Sprint Backlog
List of Features
Team Selects subset from List of Features for Sprint Release
Sprint Backlog
Product Backlog
Product Owner; Scrum Master; Development Team,
Scrum Development
Sprint Planning
Enter Sprint
Product Owner (PO) Dev Team & Scrum Master Rep from Ops Team; Build Engineer (BE) - Setup CIB Server; Setup Test Server; Setup Staging Server; Setup production server; Setup Containers in Dev machine)
Daily Code Commits Integration Tests Feedback from CIB Server
Every 24 Hours
Unit Tests Smoke Tests Task Tests Maintain Successful Build Status
1- 4 Week Dev Cycle
Continuous Integration & Build Engineering "Ceremony" Incorporated into Sprint
Product Increment (Working Feature-set)
Product Owner; Scrum Master; Development Team; Build Engineer
QA & "Done" Release
Product Increment Review (PO; Dev Team; BE)
Build & Test Deployment Package (BE may opt for Containerisation)
Sprint Review (PO; Dev Team Rep; End User Rep; BE)
"Done" Product Increment
Product Owner; Scrum Master; Development Team; Build Engineer; End User Rep

Figure 5.18: The Scrum Development Operations Model (SDOM)

The adjustments to the Scrum development cycle as illustrated by the SDOM model shown in Figure 5.18, are discussed below:

- The **Sprint Planning meeting** incorporates the Build Engineer (BE) who uses this opportunity to engage with the development team and establish familiarity with the functional specifications of the Sprint cycle. The BE is also responsible for configuring the development machines, the integration server, the test server as well as a staging server were the product increment may be tested in an environment that is as close to the production environment as possible. According to Aiello (2017) the BE may opt for adoption of a strategy referred to as **containerisation** where the production environment is simulated on development and test machines inside 'containers' that provide a development space that is independent of the underlying operating system. In this way the development team is able to develop in a truly production-like environment thereby mitigating the complexities of incompatibilities between the development, testing and deployment environments. Aiello (2017) made mention of the current technological stack that enables this strategy and suggested that the optimal tools to enable continuous integration using the strategy of containerisation are the open source tools named GitHub and Docker;

- The **Scrum stand up** meetings should include a discussion on the status of the baseline code for the evolving system. The development team should report on the frequency of their 'code commits' to the baseline system which resides on the integration/version control server. Ideally, there should at least be a single daily build that is 'triggered' by the 'code commits'. However, Aiello and Sachs (2016) caution about the undue complexity that may be added to the development overhead when there are too many builds to contend with. The reason for this concern is that the entire development team has to update changes to their local development workspace so that they always have a current version of the baseline system. If there are too many commits and builds in a short time period increases the overhead of maintaining a current (up to date) version of the system on the local development machine at all times. The benefits of engaging in a practice of continuous integration (CI) is that the baseline system is always kept in a stable state and there is minimal effort to integrate new code. The dilemma is resolved by getting team members to make a commitment to the CI practice with the undertaking that during the initial stages of development, code commits to the baseline system may be infrequent. However, as the system approaches the final stages of completion, there should be frequent code commits thereby ensuring that the final integration and testing phases do not have to deal with the complexity that may be introduced by lack of adherence to a CI strategy;

- The **testing suite** should include unit tests that consist of cyclomatic tests so that there is optimal testing of branch and looping logic. The test server that invokes pre-defined functionality tests (test driven development (TDD)) should be customised to include different stages of development. During the initial development stages, the pre-defined TDD strategy will not be feasible because of incomplete coding. However, as McConnell (1996) points out, the use of a strategy that entails smoke testing and stubs/place holders for incomplete functionality will arguably ensure that the baseline code is always in a stable state. The smoke tests are not as complex as TDD tests, but they ensure that the evolving system demonstrates basic functionality and is always in a stable state so that it runs and produces some form of output. The invocation of a smoke testing and TDD strategy enables the developers to do an internal verification that a user story or a task has been completed. This is crucial from a workflow perspective. If the team is using a Kanban Board to track the development progress, then the internal verification that the task has been successfully completed will enable the task to be labelled as *done* or *verified* from a workflow perspective, thereby freeing up the number of tasks that fall into the verification swim lane of the Kanban Board. The mechanism of the Kanban Board is that there is a pre-defined limit to the number of tasks that can be placed in a specific swim lane. However, the Scrum Board technique does not have such a restriction. Based on the evidence of the empirical data, the strategy of restricting the number of items in any specific swim lane is regarded as a good strategy because it provides a quick indication of a 'bottleneck' situation that needs to be resolved before any further development in the sprint can be undertaken;

- The role of the **Build Engineers** is to provide a testing environment that is relevant and as close to the production environment as possible. Aiello and Sachs (2016, p. 129) make reference to a "pre-flight" build where the development and integration platforms are similar to the production environment. This strategy may be viewed as a 'right shift' where the BE provides resources for the developers early in the development lifecycle so that the build quality of the evolving system can be verified before it is handed over to the operations team. Aiello (2017) makes the point that setting up of the test environment is a complicated process because the test server has to be set up so that it provides the runtime dependencies that developers were using in their local development machines. Hence the involvement of the Build Engineer as an additional role player is crucial because it arguably ensures that there is a degree of compatibility between the development, test and production environments;
- The **Product Increment** is the immediate output of a Sprint cycle. The phase traditionally referred to as 'done' is shifted to the right of the Scrum process because once a product increment has been completed, the quality of the increment has to be verified and validated prior to the allocation of a 'done' status. This verification and validation process first occurs internally between the PO, the Scrum development team and the Build Engineer during a **Product Increment Review** session. The presence of the BE is required to sort out issues that deal with the testing environment. The

documentation requirements for the product increment is also deliberated upon during this phase of the Scrum development cycle. Plans for the maintenance of the system should be incorporated into deliberations at this stage and the documentation requirements to support the maintenance activity should be identified and created by the development team;

- The BE is then responsible for developing a **deployment package** that is tested in a production-like environment. Once more a containerised version of the deployment package should be made available for Quality Assurance (QA) and user acceptance testing (UAT);
- The **Sprint Review** phase has undergone a 'right shift' due to the added layer of quality checks (product increment review) and the build engineering activity to arguably ensure that the system is in a deployable state. The Sprint Review phase is essentially a showcase of the system's functionality, usability and performance. This phase also presents an ideal opportunity for the end user to interact with the evolving system thereby providing the development team and the Product owner with an opportunity to obtain feedback regarding the system's functionality and usability. This phase is a vital inclusion in the Scrum development cycle because one of the problems identified from the study's empirical evidence is that the end users do not have ample opportunity to interact with the system and provide feedback that the developers could use to improve the usability of the system. This phase serves the purpose of re-establishing the close working relationship between the development team and representatives from the end user group. This phase is also an ideal opportunity to engage the BE on issues related to the system's performance because performance testing can be done in a production-like environment;

- The final phase of the Scrum sprint cycle is the assessment from the stakeholders that the **product increment is done** and can be made available for the build package that may be subjected to integration tests in a staging environment. Once more, a containerised version of the 'done' portion of the system is maintained and used as a baseline system onto which new elements of functionality are added during the remaining sprint cycles until the full system has been developed. The containerised 'done' version of the system is also in an immediately releasable state.

The development of SDOM is based on an interpretive analysis of empirical data where the researcher leveraged expert knowledge provided by a purposively selected group of software developers, BA's and Operations Engineers. The model represents a convergence of this knowledge that is contextualised according to the experience and expertise provided by the study's respondents. SDOM should not be seen as a definitive version of Scrum based software development practice. However, the integration of Scrum based methods for software development with methods that are deemed to be operational or infrastructure oriented should be seen as representative of a trajectory for software development methodology that takes cognisance of the operational environment in which the software will be used.

A Note on the use of Scrum as a Baseline for the Proposed Framework

Based on the analysis of qualitative data, the weighted percentage of significant words that were counted indicated that the expression *Agile methodology* accounted for 15% of the total word count and *Scrum* accounted for 10% of the total word count. These statistics are illustrated graphically on P. 187 Figure 5.4. A further investigation revealed that the transition from Waterfall methodology to an agile approach basically entailed a transition to Scrum methodology. This is illustrated in the cross-query (P. 208; Figure 5.8) that was conducted between the themes *Transition from Waterfall* and *Scrum* where it was revealed that 87.5% of the interviewees associated Agile methodology with Scrum methodology. An outcome of the analysis of the pre-questionnaire that was administered to all 16 respondents, revealed that all 16 respondents indicated that Scrum was regarded as the central methodology for software development. However, Scrum was adapted according to the organisational context to include agile methods from XP and Kanban. The overwhelming preference for Scrum was a significant factor that “cajoled” the naming convention adopted for the proposed model to contain a reference to Scrum.

5.6 Conclusion of the Qualitative Phase

The qualitative phase of the study has been designed to achieve the benefits of implementing a phenomenological approach to obtain a deeper insight into the phenomenon of agile software development as experienced by South African software practitioners in an organisational context. A total of 16 interviews were conducted, transcribed and subjected to a content analysis to enable the presentation of a rich textural description of the activity of software development. The output from this phase of the study is structured according to a socio-technical perspective resulting in 2 main models that were synthesised in the study. The first model developed, represents the 4 dimensions of organisational culture as defined by the Competing Values Framework (CVF) and its alignment with the main software development methodologies. An intended outcome of this exercise is to provide a framework that informs the transition from the Waterfall methodology to an agile methodology along the dimensions of organisational culture. The 2nd model developed represents an incursion into the operations/ build engineering phase of the development lifecycle. The model, named the Scrum Development Operations Model (SDOM) has the objective of mitigating the perceived shortcomings of agile/Scrum methodology from an operations perspective.

The final empirical phase of the study entails a quantitative exercise to determine the acceptance of SDOM, which is conducted in the next chapter.

6 QUANTITATIVE VALIDATION

As part of the methodology adopted for the study, a 2nd phase of data collection entailed a quantitative validation of the proposed Scrum Development Operations Model (SDOM). The validation exercise was conducted by a group of purposively selected expert practitioners in the domain of software development and operations. The objective of this exercise was to determine whether SDOM has an alignment with current software development practice and whether SDOM will be accepted as a useful intervention that adds value to the agile framework for software development. A peripheral objective of the quantitative acceptance exercise was to verify the 'goodness of fit' of the Theory of Acceptance of Software Development Methodology (TASDM) to the study's data. Based on the outcome of this validation process, an adjusted version of the theoretical model to determine acceptance of a software development methodology was proposed.

6.1 Introduction

The use of a quantitative approach has to be accompanied by a disclaimer to the effect that there is no intention to generalise the acceptance of SDOM to a wider population. However, the quantitative approach is used to obtain feedback from selected individuals who have a measure of familiarity and maturity with general agile software development and operations. The feedback obtained from these individuals has been operationalised via a social science behavioural model of acceptance that has a strong resonance with technology acceptance theory. The structural sequence adopted for the presentation of the quantitative phase of the study is as follows:

- A discourse on technology acceptance theory is presented with the explicit purpose of contextualising the theoretical model used in this phase of the study;
- A discussion of the questionnaire items and the process used to finalise these items;

- A description of the study's sample;
- A presentation and analysis of the study's data by making use of descriptive statistics and inferential tests of significance with regards to the data's measures of central tendency. The study's data is also subjected to tests of reliability and construct validation (via confirmatory factor analysis);
- A bivariate and regression analysis exercise is conducted to examine the relationships between the study's main constructs;
- A graphical presentation of the regression data is presented by virtue of a path analysis diagram;
- An exercise in Structured Equation Modelling is undertaken to theorise a model that has a better predictive capacity/ 'model fit' for the study's data;
- A discussion on the acceptance of SDOM, based on the quantitative data analysis;
- A discussion of the open ended comments made by the study's respondents.

6.2 The Quest for a Theoretical Lens to Determine Acceptance of a Software Development Methodology

In order to determine the acceptance of the proposed Scrum based DevOps model as a software process improvement technique, the study's design has been extended to incorporate a quantitative dimension that is underpinned by technology acceptance theory. Pfleeger (1999) makes the suggestion that the discipline of software engineering needs to draw upon social science models to further understand the adoption of technology. From this perspective, there have been several theoretical models, emanating from psychology and sociology that have been used to explain technology acceptance (Erasmus *et al.*, 2015; Venkatesh *et al.*, 2003; Venkatesh *et al.*, 2012). The majority of these models are centred on the behavioural intention (BI) to use a technology (Chau & Hu, 2002). The concept of BI is elucidated by virtue of theories such as the Theory of Planned Behaviour (TPB) (Ajzen, 1985), the Theory of Reasoned Action (TRA) (Fishbein & Ajzen, 1975), the Technology Acceptance Model (TAM) (Davis, 1985), the Unified Theory and Acceptance of Technology (UTAUT) ((Venkatesh et al., 2003) and Rogers' Diffusion of Innovations (DOI) (1983) theory. According to Kim et al. (2012) and Erasmus et al. (2015), these afore-mentioned theories are the most widely used from a technology acceptance perspective.

The use of the expression 'technology acceptance' may be misleading in the context of the current study where the focus is on software development methodology. However, reference is made to the academic defense provided by Riemenschneider and Hardgrave (2001) as well as Wallace and Sheetz (2014) who justify the use of technology acceptance theory as a proxy for theory on the acceptance of software development methodology. As a disclaimer, it should also be noted that in the annals of software engineering literature there is a dearth of guidance on the determinants of software development methodology acceptance. Riemenschneider *et al.* (2002) contend that the technology acceptance theories emanate from general theories of human behaviour. This should in all probability enable an extension of the domain of application of these theories to a realm that is beyond just technology adoption and to include the intention to use a software development methodology. This claim was backed up with the presentation of empirical evidence that attested to the validity of technology acceptance models to predict acceptance of software development methodology. The afore-mentioned claim is based on a study by Riemenschneider *et al.* (2002) that entailed the gathering of survey data from software practitioners on the determinants of software development methodology acceptance. The study examined the significance of technology acceptance constructs in ascertaining the adoption and acceptance of software development methodology. The outcome of this study is that the technology acceptance constructs are all valid predictors of acceptance of software development methodology. The construct of Perceived Usefulness (PU) was found to be the most reliable predictor of intention to use a software development methodology. The reliability of technology acceptance constructs as a predictor of intention to use a methodology have been confirmed in Hardgrave and Johnson (2003), Johnson (1999), Templeton and Byrd (2003), Chan and Thong (2009) and Wallace and Sheetz (2014). Chan and Thong (2009) do however, caution that the acceptance of a methodological approach as opposed to technology adoption needs to be examined with theoretical models that provide a "lens" that covers not only the technical factors, but also caters for the non-technological factors such as individual and organisational characteristics.

While TAM provides implicit coverage of the non-technological factors that influences adoption behaviour, TAM2, UTAUT and DOI incorporate constructs that make explicit reference to the social and organizational domain. Based on the preceding argument, the current study engages in an overview coverage of the TAM2, UTAUT and DOI theoretical models with the intention of identifying a viable academic underpinning that will guide the collection of empirical evidence on the acceptance of the proposed Scrum based DevOps model for software development.

6.2.1 Acceptance Theory

The theoretical underpinnings of TAM are centered on the psychological factors of perceived usefulness (PU) and the perceived ease of use (PEOU) of using a technology. Upon closer scrutiny of the data emanating from studies that tested TAM theory, Venkatesh and Davis (2000) posited that empirical studies have confirmed PU as a stronger and more reliable determinant of usage intention than PEOU. However, as Edmunds *et al.* (2012, p. 4) point out, the "…interaction between technology and its acceptance for use is multi-faceted" and the two primary constructs of PU and PEOU are not sufficient to capture the essence of this interaction. A significant limitation of TAM is the inadequate focus on the social context in which the technology is being used. There is a lack of reference to the social context from the perspectives of general use of technology in an informal setting (Evans *et al.*, 2014) as well as in a formalised organisational setting (Legris et al., 2003). In order to improve the predictive capacity of TAM so that there is cognisance of the social and organisational context in which technology is used, Venkatesh and Davis (2000) proposed the TAM2 model illustrated in Figure 6.1.

According to Venkatesh and Davis (2000) the additional constructs of the TAM2 (illustrated in Figure 6.1) improved the predictive capacity of the TAM by approximately 20% (TAM accounted for only 40% of the variance in technology acceptance whereas TAM2 was able to account for almost 60% of the variance). The additional constructs are classified by Venkatesh and Davis as the Social Influence Processes and the Cognitive Instrumental Processes. An overview of the TAM2 model together with the constructs is provided below:

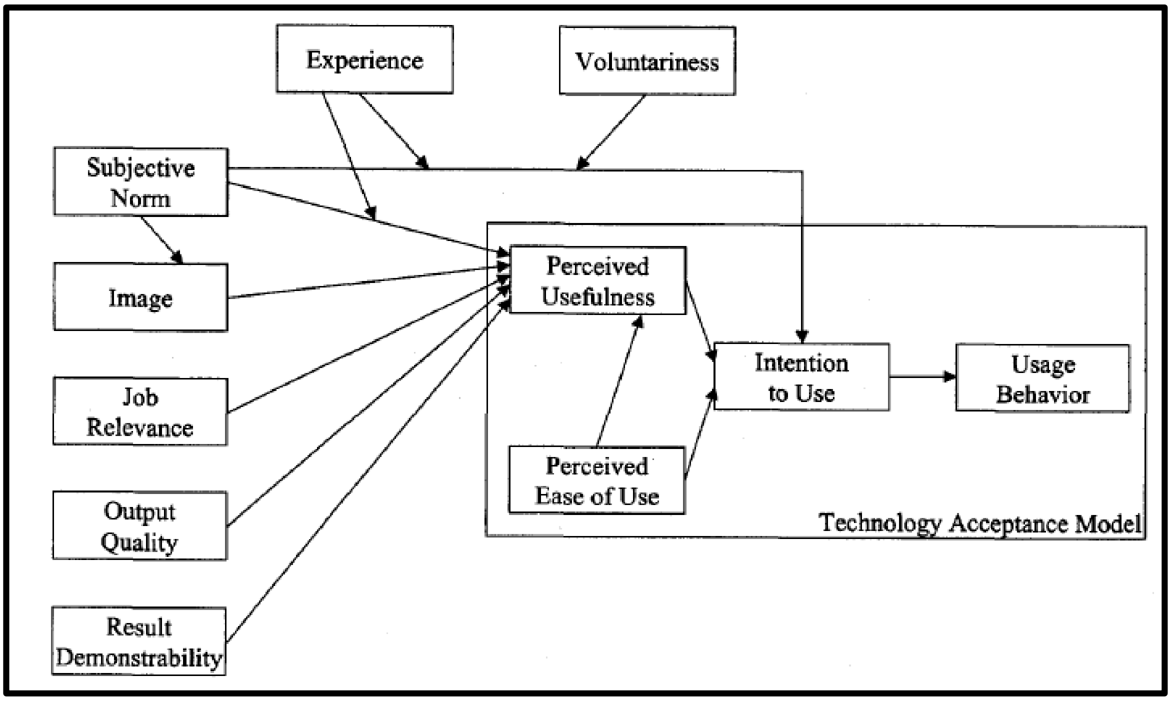


Figure 6.1: TAM2 model proposed in Venkatesh and Davis (2000)

The Social Influence Processes

- Subjective Norm: A construct (with roots in the Theory of Reasoned Action and the Theory of Planned Behaviour) that is defined as the degree to which an individual is of the opinion that usage of the technology is endorsed by 'important others' or "people of influence" within the individual's context of use of the technology;

- Image: A construct (with roots in the Diffusion of Innovation theory) that is defined as the degree to which the use of an innovation enhances one's status in a social system. From a work-oriented perspective, this construct is a reference to an individual's perception that by using a technology, it will lead to improvements in that individual's job performance by virtue of an enhancement in the individual's image in the work environment.

Cognitive Instrumental Processes

- *Job relevance*: A reference to the degree to which a user perceives a technology to be applicable to that individual's job. This judgement is based on the alignment between the functionality offered by the technology and the functionality required by a job. If an individual perceives a technology to be not relevant to a job situation, then that technology is discarded from that individual's set of options for consideration with regards to completing a job;
- *Output quality:* A reference to more than just a capacity of the technology to handle a specific processing requirement, but a judgement on how well the technology is able to perform a job task. In contrast to job relevance, if the technology has some relevance but does not contribute optimally towards job completion, it is still considered as a viable option;
- *Result Demonstrability*: A reference to the degree to which an individual attributes productivity in their job performance to the use of the technology. However, if the role of the technology in enhancing job performance is obscure, then an individual is less likely to continue using the technology.

The essence of the TAM2 model is embodied by the subjective norm construct which according to Venkatesh and Davis (2000), has a significant moderating influence on PU. It should be noted that the absence of subjective norm from TAM has been identified as a limitation of TAM's predictive capacity. The preceding claim is corroborated in Schepers and Wetzels (2007) where a meta-analysis of 51 articles containing 63 empirical studies of TAM as a predictor of the intention to use a technology confirmed the influence of subjective norm on PU and PEOU. The moderating effect of subjective norm is that it factors in the organizational context in which PEOU and PU can be evaluated, thereby establishing a tangible link between the organisational culture and the acceptance and use of technology.

Aligned to an imperative to develop a technology acceptance model that has an improved predictive capacity, Venkatesh *et al.* (2003) conducted a review of 8 competing theoretical models that identified up to 7 constructs used to determine the acceptance of technology. The outcome of the study was that only 4 of these constructs had a significant influence on user acceptance of technology and usage behaviour. These UTAUT constructs (illustrated in Figure 6.2), are elaborated in Venkatesh *et al.* (2003). An overview of these constructs is presented below, together with a comment regarding an alignment with the constructs from TAM and TAM2.

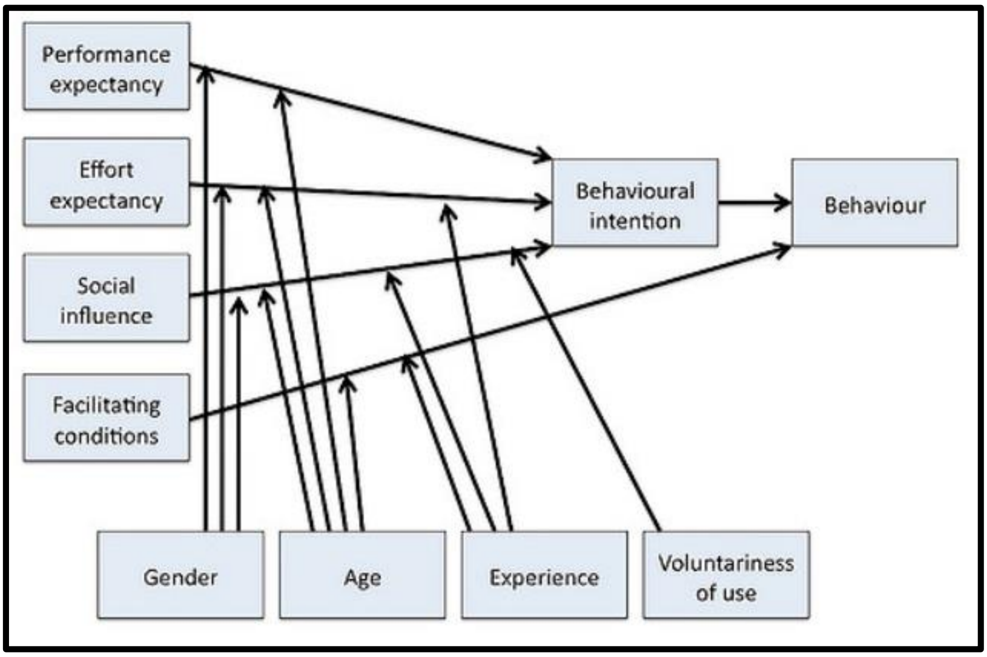


Figure 6.2: UTAUT model proposed in Venkatesh *et al.* (2003)

These UTAUT constructs (illustrated in Figure 6.2), are elaborated in Venkatesh *et al.* (2003). An overview of these constructs is presented below, together with a comment regarding an alignment with the constructs from TAM and TAM2.

- *Performance expectancy*: The degree to which an individual believes that system usage will enhance his/her job performance. This construct resonates quite well with *job relevance* and *output quality* from TAM2 and perceived usefulness from TAM (Dwivedi *et al.*, 2011; Venkatesh *et al.*, 2003);
- *Effort expectancy*: A measure of the ease with which a system may be used. This construct is conceptually identical to the PEOU construct contained in TAM and TAM2 (Dwivedi *et al.*, 2011; Venkatesh *et al.*, 2003);
- *Social influence*: The degree to which an individual is of the opinion that system usage is endorsed by 'important others' or 'people of influence' in an organisational setting. This construct has a strong alignment with the *subjective norm* construct found in TAM2. It also provides a conduit through which organisational culture may be factored into an individual's intention to use a new technology;
- *Facilitating conditions*: A reference to the level of organisational support and the capacity of the infrastructure to facilitate use of the new technology. This construct is an objective indictment on the resources available within the organisation as well as the user's self-efficacy in handling the cognitive demands of using the system. This construct does not have a direct alignment to any of the constructs from TAM and TAM2.

UTAUT also posits that the influence of the above-listed determinants of behavioural intention to use technology is moderated by the gender, experience and age of the user. Voluntariness also has a mediating effect on social influence. The UTAUT model is validated in Venkatesh *et al.* (2003) who make the claim that

UTAUT is able to account for 70% of the variance in intention to use a technology. This is a substantial improvement to the TAM model which accounted for just 40% of the variance in usage intention (Legris *et al.*, 2003). The main reason for this improvement is the inclusion of socially oriented constructs in the UAUT model, as opposed to an exclusive focus on the technology aspects of usage intention as embodied by the TAM. However, UTAUT only exhibits a 10% improvement on the predictive capacity of TAM2, thereby rendering both these models as viable options to study technology acceptance in a social and organisational context.

Another highly influential acceptance model that seeks to explain the acceptance or rejection of an innovation is the Diffusion of Innovation Theory (Miranda *et al.*, 2014). Everett Rogers, a sociologist, developed the concept of innovation, which he regarded as any object, idea, technology, or practice that is new (Rogers, 1983). Rogers identified several intrinsic characteristics of innovation that influence an individual's decision to adopt or reject an innovation. These characteristics became the basis for Rogers' Diffusion of Innovations (DOI) theory that is widely used in the information technology (IT) field to study adoption of technological innovations (Lyytinen & Damsgaard, 2001; Pozzebon *et al.*, 2014). Chang (2010), stressed the relevance of DOI theory by asserting that innovative products or ideas have a widespread influence on society and the adoption behaviour of the wide range of stakeholders is best understood by leveraging theoretical models such as DOI.

In the context of the current study, the proposed Scrum based software process model incorporates a wide range of active participants who are required to collaborate on a more frequent basis to facilitate the delivery of a successful software system. The integration of the wider range of stakeholders such as business representatives, the development team, the end user and the operations engineer into the actual development process may be viewed as innovative, especially from a DevOps perspective. It is within this context that Rogers' DOI theory may be seen as a viable theoretical framework.

According to Rogers (1983), there are 5 intrinsic characteristics of innovations that influence an individual's decision to adopt or reject an innovation. These factors together with a description of each factor are presented in Table 6.1.

Table 6.1: Roger's (1983) Diffusion of Innovation Factors

Factor	Description
Relative Advantage	The improvement offered by a current innovation over its predecessor
Compatibility	The level of compatibility that an innovation has in enhancing the prospect of being assimilated into an individual's life
Complexity or Simplicity	An individual's perception of how difficult it is to use an innovation; a determinant of whether the individual is likely to use the innovation
Trialability	The ease with which an innovation may be subjected to experimentation; The reasoning here is that if it is easy to test an innovation, then it will in all likelihood be easier to use
Observability	The extent that an innovation is visible to others. An innovation that is more visible will drive communication among the individual's peers and personal networks and will in turn create more positive or negative reactions.

6.2.2 A Unified Theory of Acceptance for Software Development Methodology

Although there are various viable theoretical models that may be used to investigate the phenomenon of technology acceptance, all of these models have a congruous structure. The independent constructs resonate with the following axial classifications that provide a linkage between the various acceptance theory models:

- *Technical*: Perceived usefulness (TAM) or performance expectancy (UTAUT); Perceived ease of use (TAM) or effort expectancy (UTAUT) or complexity (DOI);

- *Social*: Social Influence (UAUT) or subjective norm (TAM2) or observability (DOI);
- *Socio-technical*: Facilitating conditions (UTAUT) or Compatibility (DOI).

Having completed a review of the popular technology acceptance theory, reference is drawn once more to a study by Riemenschneider *et al.* (2002) in order to identify a cogent set of constructs that are reliable predictors of the intention to use a software development methodology. The objective of the Riemenschneider *et al.* (2002) study was to obtain empirical evidence to establish whether there is clear distinction between factors that influence the adoption of a technology as opposed to the adoption of a software development methodology. A total of 128 software practitioners were surveyed on their intentions to use a software development methodology. The main constructs from technology acceptance theory were used to structure the questionnaire that was used in the study. The questionnaire, which was subjected to internal validity tests, consisted of standardised questions that are used to measure technology acceptance (validated in the original technology acceptance theories). A significant outcome of this exercise is that voluntariness, perceived usefulness (PU), compatibility and subjective norm were found to be the only reliable predictors of behavioural intention to use a software development methodology. An analysis of these factors in the context of a behavioural response to the adoption of a software development methodology reveals that:

- *Voluntariness* is expected to have a significant influence on an individual's decision to adopt a software development methodology in an organsational context because once an organisational mandate is issued, then employees are required to make an effort to comply. Also a change to a new methodology is quite radical requiring complete transition to the mandated methodology. However, voluntariness is not the only driver of the decision to accept a software development methodology;

- The most significant construct that measures acceptance of a new software development methodology is *Perceived Usefulness* (PU). This outcome is aligned to similar findings by Venkatesh *et al.* (2003) in the context of technology adoption as well as Dyba *et al.* (2004) in the context of software developers' intentions to adopt a new software process improvement initiative. This outcome is explained from a behavioural perspective within an organisational context where employees have an innate desire to achieve optimal performance so that they can benefit from performance based reward structures. The imperative to engage with methodologies that enhance productivity and quality is the biggest driver of the behavioral intention to adopt a methodology;
- The greater the *compatibility* a new methodology has with current work practice, the more likely it is that employees will form intentions to use it, especially if there is a perception that it will enhance their productivity and quality of work (PU). The corollary situation also applies in the sense that if a new methodology deviates substantially from current work practice and there is a perception that it may not be useful, then software developers are less likely to adopt the methodology. In the context of the current study, this is an important construct to measure because the proposed model of software development has been structured according to a Scrum-based development approach that has been empirically endorsed as a useful methodology;
- The final determinant of software developers' intention to accept a new methodology is *subjective norm*, which is intrinsically linked to the culture within an organisation. If a mandated new methodology is perceived to be useful and compatible with current work practice, software developers may still avoid using it if there is also a perception that fellow employees and supervisors think that they should not be using it. This construct has a strong resonance with

> the Group Culture dimension of the Competing Values Framework that explains the different types of organisational culture. As much as developers may perceive a new methodology to be useful, they are also driven by the desire to uphold key working relationships and preserve the existing 'social order'. The use of a new methodology in such an instance may be advocated incrementally in a manner that is minimally disruptive.

The outcome of the study by Riemenschneider *et al.* (2002) has a direct influence on the operationalisation of the concept of 'acceptance' in the context of software development methodology as well as the context of the current study. The review of the various technology acceptance models provided an insight into the general constructs used within the domain of information systems research. However, the Riemenschneider *et al.* study provides a focused view of these constructs from a software development methodology perspective. This knowledge played a pivotal role in the researcher's decision to use an adapted version of this model to underpin the current study's imperative to ascertain acceptance by software practitioners of the proposed Scrum based model for software development. The adaptation made to the original theory is in reference to the construct of *voluntariness* which does not apply to the context of the current study. A global study by Ahmad *et al.* (2016) to determine software practitioners' acceptance of Kanban methodology as a viable methodology for software development implemented an adapted version of the Riemenschneider *et al.* model. The *voluntariness* construct which was not deemed to be appropriate for the objectives of the study, was replaced with *perceived organisational support* which was found to be a significant predictor of software practitioners' intention to adopt a Kanban oriented approach to software development. Based on the narrative presented, a conceptual model (illustrated in Figure 6.3) of the Theory of Acceptance of Software Development Methodology (TASDM) is used for the purpose of the quantitative dimension of the current study.

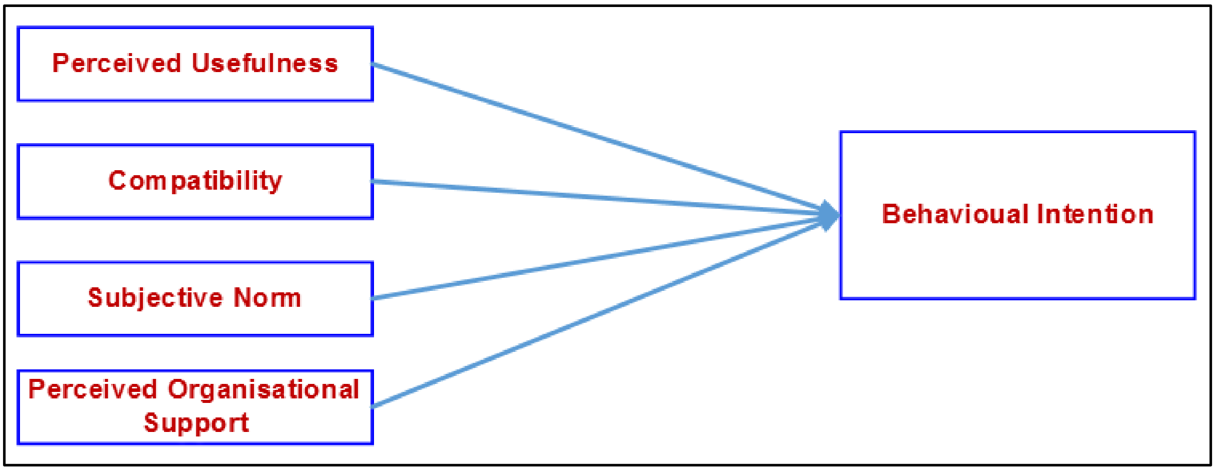


Figure 6.3: Adaptation of the Riemenschneider *et al.* (2002) Theory of Acceptance of Software Development Methodology (TASDM)

The constructs of Perceived Usefulness (PU), Compatibility (CO), Subjective Norm (SN) and Perceived Organisational Support (POS) will be used to operationalise the acceptance of the proposed SDOM for software development. The dependent variable Behavioural Intention (BI), will be used to measure the intention of software practitioners to implement the proposed model if an opportunity arises.

6.3 The Data Collection Preparatory Phase

The empirical phase of the study consists of a survey to establish software practitioners' acceptance of SDOM. The data collection instrument that is used is a questionnaire (see Appendix D). The sample for this phase of the study has been purposively selected to facilitate an alignment with the sample used in the first (qualitative) phase of the study. The questionnaire design, review, ethical considerations and the sample used in the study is discussed in the subsequent sections.

6.3.1 The Questionnaire Design

The questionnaire has been designed to align with the main constructs of the TASDM and consists of a set of pre-formulated questions that has been validated as reliable measures of software development methodology acceptance in Riemenschneider *et al.* (2002) and Ahmad *et al.* (2016). The constructs of Perceived Usefulness, Compatibility and Subjective Norm are standardised questionnaire items that have been used in TAM, TAM2, UTAUT and DOI based studies. According to Sekaran and Bougie (2010), a questionnaire is an efficient data collection mechanism when the researcher has good knowledge of the main variables of interest and the items used to measure these variables have been validated in previous studies. The questionnaire has been designed to consist primarily of close-ended, Likert scale type of questions containing 5 descriptors that ranged from 'strongly agree' to 'strongly disagree'. The explicit purpose behind the use of this design strategy was to enable the respondents to make intuitive decisions regarding the alternative responses and also to enable the researcher to easily code the data for subsequent analysis. This approach is aligned to similar strategies used for acceptance/adoption based research (e.g. Riemenschneider *et al.*, 2002; Venkatesh & Davis, 2000; Venkatesh *et al.*, 2003). Respondents were however, provided with a section of the questionnaire where they could provide an open-ended response in the form of comments or suggestions about SDOM.

The layout of the questionnaire is shown in Table 6.2.

Table 6.2: Layout of the Questionnaire

Section	Topic of Section	No of Questions
	Demographic & Background Information	7
1	Perceived Usefulness (PU) of the Proposed SDOM	6
2	Compatibility of the Proposed SDOM	3
3	Subjective Norm/Social Factors that Influence the use of the Proposed SDOM	3
4	Perceived Organisational Support for the use of SDOM	2
5	Behavioural Intention to use SDOM	2
6	Comments/Feedback/Suggestions on SDOM	1

The main sections in the questionnaire are discussed below:

Introduction

The Introduction section is used to establish a context for the study as well as explain the main outcomes that relate to the development of SDOM from the first phase of the study. An illustration of SDOM is provided in the questionnaire for quick reference. Respondents are also directed to view a detailed narrative on SDOM that has been made accessible via the study's website. The website is accessible at: http://143.128.146.30/SDOM/ScrumOps/SDOMIntro.aspx

Demographic and Background Information

Respondents were required to provide their names and surnames as well as details regarding the type of organisation that they belonged to, the capacity that the respondent served in the organisation and the number of years of experience in the domain of software development. All respondents were however, informed of the voluntary nature of their participation and the measures that will be taken to assure their anonymity and the confidentiality of the data that is provided.

Perceived Usefulness (PU) of SDOM

The questions are phrased in a manner that enables the attainment of a measure of the usefulness that SDOM may provide for software practitioners working in an organisational setting.

Compatibility (CO) of SDOM

The questions are phrased to obtain knowledge of the compatibility that SDOM has with the current work-based practices of software practitioners.

Subjective Norm (SN) that Influence the Use of SDOM

The questions are phrased to ascertain whether the use of SDOM will be acknowledged as a progressive intervention by colleagues and people of influence to the software practitioners.

Perceived Organisational Support (OS) for the Use of SDOM

The questions are aligned to those used in the Ahmad *et al.* (2016) study and have been phrased to determine the perceived resource-based support that the organisation's management will provide for the use of SDOM.

Behavioural Intention (BI) to Use SDOM

The construct of BI is the only dependent variable and the questions are phrased to ascertain whether the respondents have a preference for the use of SDOM if an opportunity arises.

Comments/Suggestions

The final section of the questionnaire provides the respondents with an open-ended forum to document their comments or suggestions about SDOM. It is envisaged that responses from this section will be crucial to help identify aspects of SDOM that could be improved upon or possibly require a re-engineering intervention.

6.3.2 The Pilot Study and Ethical Clearance

The questionnaire was piloted with 2 academics from the Discipline of Information Systems & Technology at University of KwaZulu-Natal (UKZN) and one industry professional who has 7 years of experience as a general software practitioner and 5 years of experience with Scrum oriented software development.

The main point of contention during deliberations in the piloting phase involved the syntax and the semantics of the PU section of the questionnaire. Four of the six questions in this section made use of a personal pronoun. The panel was of the opinion that the choice of pronouns should be changed because SDOM is a model that has relevance to software practitioners in the context of their involvement in software development as part of a team of developers who work in an organisation. The original questionnaire made a reference to the respondents with regards to the influence that SDOM may have if they acted individually in a personal capacity. This wording of the question was changed slightly so that the semantics were aligned to the respondent's perception of SDOM in the context of perceived usefulness for a software development team functioning in an organisational setting. The panel was of the opinion that the remainder of the questions were clear and concise enabling easy comprehension. The pilot panel was also of the opinion that the respondents to the survey were provided with adequate documentation to enable full comprehension of the study's context and the research objectives. The data collected in the pilot study was not used as part of the data corpus during analysis of the quantitative data.

Ethical clearance for the quantitative phase of the study was obtained from the Committee for Research Ethics at UKZN (see Appendix E).

6.3.3 The Sample Used for the Quantitative Phase

The sample for the current phase of the study consists of members who were purposively identified and invited to participate in the qualitative phase (first) of the study. This purposive approach is deemed as necessary because members of the sample from the current phase of the study were required to meet the following criteria:

1. Each member of the sample must have been eligible to participate in the qualitative phase of the study. The criteria used in the qualitative phase of the study is that participants of the study must have at least 5 years of experience as a software developer and at least 2 years of involvement with agile software development methodology.
2. Members from the sample must have some familiarity with the context of the current study. This would have been achieved during the researcher's initial contact with the prospective subjects of the study during which time the objectives of the study were explained to them.

An additional requirement that has been necessitated by the type of model that was developed in the study was to include practitioners who have knowledge or expertise in the domain of Build Engineering. During the first phase of the data collection, it became apparent that many of the respondents were very much aware of the deployment requirements and the need for Build Engineering expertise to be included into the Scrum development methodology. However, the role played by BE was not well defined and accorded the recognition that was deserving of this responsibility. From an empirical perspective, the first phase of the study did include 3 members of the sample who have experience or expertise in the domain of Build Engineering. In order to mitigate for the relative lack of representation from the Build Engineering domain during the first phase of data collection, an effort was made to include representation by 5 additional practitioners who have experience in the Build Engineering domain.

The sample group size for the quantitative component was computed to be 45. This included:

- All 16 interviewees from the first phase of the study;
- A further 24 participants from the first phase of the study who indicated their willingness to contribute, but did not from part of the interview cohort;
- Five representatives from the Build Engineering domain.

6.4 The Quantitative Data Presentation

A total of 45 questionnaires were e-mailed to the members of the sample. Forty completed questionnaires were returned yielding a response rate of 88%. The responses to the Likert Scale questions were analysed by making use of the Statistical Package for the Social Sciences (SPSS) software. The data was initially coded by assigning the numbers from 1 to 40 to each of the returned questionnaires. The individual questions were given variable names and numerical values ranging from 1 to 5 were used to capture the actual response indicated in the questionnaire. In terms of the extremes, a value of **5** was assigned to the Likert scale option of '**strongly agree**' and a value of **1** was assigned to the Likert scale option of '**strongly disagree**'. A value of **3** was assigned to the Likert Scale option of '**neutral**'.

6.4.1 Reliability Testing

As suggested by Gliem and Gliem (2003), when Likert scales are used, it becomes imperative to compute and report the Cronbach's alpha co-efficient to establish the reliability of the questionnaire. According to Sekaran and Bougie (2010, p. 324) the reliability test is used to determine "…how well the items measuring a concept hang together as a set". In general, a Cronbach Alpha coefficient value that is less than 0.6 is regarded as 'poor', indicating that the set of questions do not provide a reliable measure of a specific construct. Values that are in the range from 0.7 to 1 are reflective of a reliable measure of a specific construct.

The reliability of the constructs used to measure acceptance of SDOM is presented in Table 6.3.

Table 6.3: Cronbach Alpha Coefficient Values

Construct	No of Likert Scale Items	Cronbach's alpha
Perceived Usefulness (PU) of the Proposed SDOM	**6** (abbreviated as PU1 to PU6)	0.752
Compatibility of the Proposed SDOM	**3** (abbreviated as Comp1 to Comp3)	0.737
Subjective Norm/Social Factors that Influence the use of the Proposed SDOM	**3** **(**abbreviated as SN1 to SN3)	0.775
Perceived Organisational Support for the use of SDOM	**2** (abbreviated as OrgSupp1 to OrgSupp2)	0.900
Behavioural Intention to use SDOM	**2** (abbreviated as BI1 to BI2)	0.704

As can be observed in Table 6.3, all the Cronbach's alpha values are in excess of 0.7, indicative of a data set that may be seen as a reliable measure of the acceptance of SDOM.

6.4.2 Quantitative Data Preparation

Confirmatory Factor Analysis (CFA) was used to confirm that the latent variables identified by the Likert scale items of the questionnaire are aligned to the TASDM. In order to obtain an individual score for each of the major factors/latent variables identified in the CFA as well as the TASDM, averages of the individual Likert scale items for each latent variable was computed. This procedure of collapsing several Likert Scale items into a single variable by computing an average value is rather controversial in the annals of statistical scholarship (see Allen & Seaman, 2007; Boone & Boone, 2012; Jamieson, 2004; Norman, 2010). The reason for the controversy is that the process entails a conversion from ordinal data (the original Likert scale items) into interval data (the average values). Norman (2010) does however, provide comprehensive evidence to verify the validity of this approach. Boone and Boone (2012) explain that a possible reason for the controversy is the lack of clarity between Likert scale items and a Likert scale measure. A Likert scale measure alludes to a latent variable that is operationalised by many Likert scale questionnaire items. When these items are combined into a composite value using techniques such as the mean computation or a summation (also suggested as a data reduction technique in Sekaran and Bougie (2010, p. 311)), then the resulting Likert scale value may be treated as interval data (also confirmed in Brown (2013)). In the context of the current study, the Likert scale items are cohesively aligned to the main constructs of the TASDM (confirmed by the CFA results) thereby enabling these individual Likert scale items to be coalesced into the 4 broad Likert scale measures of PU, CO, SN and OS.

The next aspect of 'statistical controversy' concerns the assumptions that underlie many of the statistical tests. These are the assumptions of randomness and normality of data.

The Issue of Randomness

The intention of the quantitative data analysis and presentation section is to obtain quantified knowledge of the acceptance of SDOM by the *respondents* in the study. As a disclaimer, the statistical analysis conducted does not represent an attempt to extrapolate the results to a wider population. The sampling strategy used in the current study is purposive sampling, thereby violating the assumption of randomness that is a pre-requisite for inferential statistical analysis. The objective of the statistical analysis exercise is to obtain a summative overview of the data. This imperative will be achieved by making use of univariate and multivariate analysis techniques. Both the afore-mentioned techniques will implement a hypothesis testing strategy to answer questions about the statistical significance of the relationships between:

- the main constructs of the study and the statistical measures of central tendency such as the mean, median and the mode of the data (univariate);
- the main constructs of study by implementing correlation statistical analysis techniques (multivariate).

The Issue of Normality

Statistical data analysis is classified according to 2 main techniques. These are parametric and non-parametric tests. The difference is that parametric tests are traditionally based on the assumption that the data is normal (normally distributed) whereas, non-parametric tests do not make any assumptions about the distribution of the data (Agresti, 2018). However, the enforcement of rigid rules that guide the choice of statistical tests has also become a source of controversy (Norman, 2010). The accepted heuristic in the annals of statistical theory is that parametric statistical tests should only be conducted when there is a large sample size and the data is normally distributed. If the assumption of normality is not met, then the data should be subjected to non-parametric statistical tests which provide a more robust alternative for data analysis. These heuristics have however, been subjected to extensive scrutiny in various simulation exercises where the results did not corroborate the heuristics (e.g. Kitchenham *et al.*, 2017; Norman, 2010). There are many instances where parametric tests provide a more robust analysis alternative to non-parametric tests including situations where there is a small sample size and the data does not conform to a normal distribution. Added to this mix of deliberations is the Central Limit Theorem which states that in a sample where the sample size exceeds 30, the distribution of the sample means will be approximately normal (McClave *et al.*, 2012). Hoskin (2012) does however, provide some guidance on the choice of statistical tests by suggesting that the parametric route should be taken if the sample size is greater than 30 ($n>30$) because parametric tests are easier to interpret and have greater statistical power than the equivalent non-parametric tests. However, if the data displays a significant deviation from the condition of normality, then there is no option but to make use of non-parametric tests.

The preceding discourse provides a rationale for attempting to gravitate the choice of strategy for the statistical analysis in the current study towards the parametric domain of statistical analysis. As a disclaimer for this approach, reference is drawn to the comments made in a highly cited article by Norman (2010, p. 7) that:

> *Parametric statistics can be used with Likert data, with small sample sizes, with unequal variances, and with non-normal distributions, with no fear of coming to the wrong conclusion.*

However, an intervention to align the analysis with the expectations of the 'statistical purists' will be made by resorting to non-parametric methods if the condition of normality is not met. A final word on the issue of controversy with regards to statistical testing is accorded to Wilkinson (1999, p. 601) who makes the point that " …there is no substitute for common sense" and a researcher should be guided by heuristics that determine whether the statistical outcome makes sense and the procedure used is appropriate for the type of study being undertaken.

6.4.3 Construct Validity Testing

According to Sekaran and Bougie (2010, p. 160) construct validity is a strategy used to determine how well the results obtained from a study "fit" the theory that underpins the data collection and analysis. Remler and Van Ryzin (2011) refine the concept of construct validity by suggesting that the main constructs or variables in a study should converge (correlate) with variables that are predicted by the theory. Also, the main constructs of the study are not expected to have a significant relationship with other variables of the study where this relationship is not aligned to the theoretical model, a concept referred to as discriminant validity. In order to verify the convergent and divergent validity of the study's data, the multivariate technique named Factor Analysis is used. There are 2 types of factor analysis techniques that may be used. In the case where a study does not have an a-priori theoretical model, then Exploratory Factor

Analysis is used to enable the data patterns to dictate the theoretical model. However, when a study is underpinned by a theoretical model, as is the case for the current study, then the ideal preference is for Confirmatory Factor Analysis (CFA) where the theory is used to find the best 'fit' for the data. According to Williams *et al.* (2010), the norm is that sample sizes greater than 300 enhance the reliability of the CFA exercise. However, in cases where each factor is defined by several variables, the sample size can be relatively small, and in cases where the normality of the data is not severely compromised, sample sizes of at least 40 are suggested. In the context of the current study's data parameters (n=40, assumption of normality is based on the discussion in Section 6.4.4) the pre-requisites for CFA are minimally achieved, thereby subjecting the results of the CFA exercise to the disclaimer that the lack of a better sample size may compromise the validity of the analysis. The CFA exercise is however a very good strategy for the researcher to obtain overview knowledge that the data pattern has some form of alignment to the theory. The CFA analysis was conducted by making use of the Analysis of the Moment Structures (AMOS) plug-in software for the SPSS package. An illustration of the CFA model produced by AMOS for the current study's data is shown in Figure 6.4.

The factor loadings, displayed as values that range from 0 to 1, are the main indicators of convergent and divergent validity. As an indication of good convergent validity, there should be high (>0.5) factor loadings from the main constructs of the theory (represented as ellipses in Figure 6.4) to the observed variables (questionnaire items) represented as rectangles in Figure 6.4. As an indication of good discriminant validity, there should be low (<0.5) covariance factor loadings between the main constructs of the model.

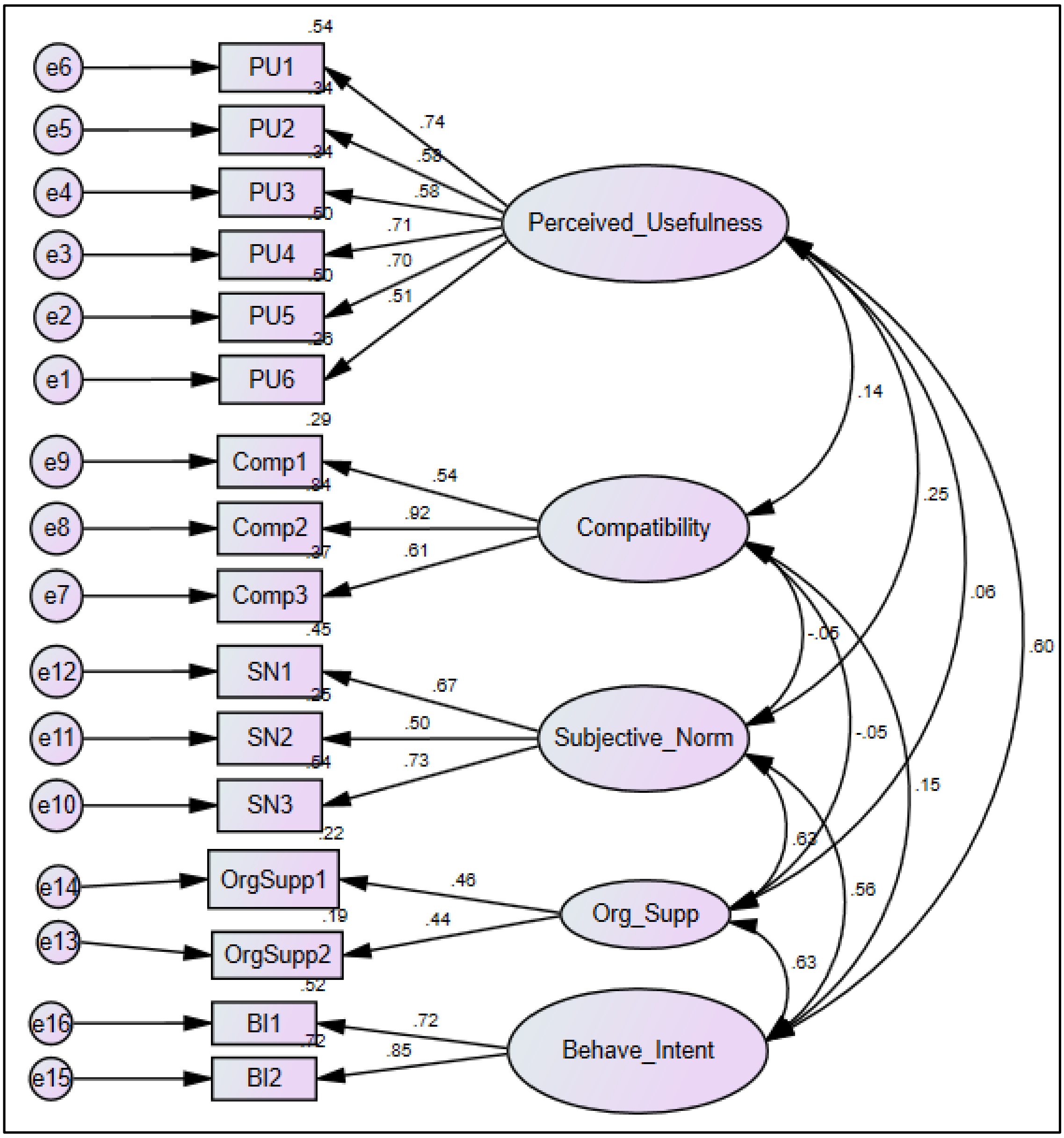


Figure 6.4: Confirmatory Factor Analysis of the TASDM

An analysis of the outcome of the CFA exercise reveals the following information:

- Four of the five main constructs show good convergent validity. The exception being the Organisational Support (OS) construct where the convergent validity is less than 0.5;

- The constructs of Perceived Usefulness (PU), Subjective Norm (SN) and OS have poor discriminant validity relative to Behavioral Intention (BI) indicated by a high covariance values (>0.5). This outcome is not entirely unexpected from a PU and SN perspective because a respondent who perceives SDOM positively with regards to these constructs will in all likelihood have a strong intention (BI) to use the model (a hypothesis that is confirmed in Riemenschneider *et al.* (2002) and resonates with general acceptance based theoretical models). The covariance between OS and BI will be subjected to further analysis;
- The constructs of SN and OS have poor discriminant validity, indicated by a high covariance value of 0.56. This result will be subjected to further analysis;
- In terms of the overall 'data fit' to the theoretical model, also referred to as the Goodness of Fit (GFI) index, it is reported in Cheung and Rensvold (2002) that the main measures of GFI are the Tucker Lewis Index (TLI) and the Comparative Fit Index (CFI) which should both be in the range from 0.9 to 1. The TLI and CFI values reported as part of the CFA output are both outside this range (TLI=0.736; CFI=0.86) suggesting that the study's data does not have an optimal fit with the theoretical model. This interpretation may however, be compromised because of the issue of a low sample size. Based on the subsequent analysis of the data, an attempt will be made to find a 'better fitting' theoretical model for the study's data.

6.4.4 Quantitative Data Analysis

The data analysis will be introduced by presenting a graphical representation in the form of frequency charts that will be used to illustrate the responses received from the sample with regards to their perception of the usefulness, compatibility, subjective norm and organisational support towards SDOM. In order to have a clear foundation for the subsequent data analysis, the study's data is subjected to a test of normality. According to Lott (2015), 2 prominent tests for normality are the Kolmogorov-Smirnov and the Shapiro-Wilk tests. Both these tests specify a null hypothesis that the data is not significantly different from a normal distribution. The main output from these tests is a p-value that provides a probability indicator attesting to whether the sample is normal. A p-value *greater* than 0.05 (95% confidence) is usually used as a condition to accept the null hypothesis that the sample has a normal distribution. The data representing the main constructs from TASDM was subjected to the Normality tests that are available in the SPSS package. The tests that were conducted are the Kolmogorov-Smirnov (KS) and Shapiro-Wilk (SW) tests of normality. The KS test has however, been criticised for being less accurate than the SW test especially when it comes to the handling of extreme values in the data (Ghasemi & Zahediasl, 2012; Steinskog *et al.*, 2007). Also, the SW test has greater statistical power when it comes to handling data from small sample sizes (n<50). Based on the preceding argument, the SW test for normality has been adopted as the main measure of normality.

The results from the SW test for normality are illustrated in Table 6.4.

Table 6.4: SW Tests of Normality for the Constructs from TASDM

	Shapiro-Wilk		
	Statistic	df	Sig.
Perceived Usefulness (PU)	0.909	40	0.004
Compatibility (CO)	0.935	40	0.024
Subjective Norm (SN)	0.954	40	0.096
Org Support (OS)	0.918	40	0.007
Behavioural Intention (BI)	0.872	40	0.007

As can be observed in Table 6.4 the constructs of PU, CO, OS and BI all fail the SW test for normality (null hypothesis rejected, p<0.05). However, SN passes the test for normality (null hypothesis accepted, p>0.05). Based on 'pure' statistical theory, the implication of the rejection of the assumption of normality is that non-parametric testing should be the default strategy. The Central Limit Theorem does however, introduce an element of doubt because the sample of 40 also renders the parametric approach as a viable alternative. Kim (2013) provides some advice in handling a dilemma of this sort by suggesting that the skewness (measure of asymmetry) and kurtosis (measure of pointiness) may also be used as indicators of normality. The decision to opt for parametric or non-parametric testing will be taken on a case by case basis that depends on the shape of the data as rendered by the frequency graph illustrations. In the case of the parametric tests, the mean will be used as the indicator of central tendency of the data. In the case of the non-parametric tests, the median will be used as a measure of central tendency.

Based on the guidance provided in Boone and Boone (2012), the one sample t-test (parametric) and the Wilcoxon one-sample signed rank test (non-parametric) will be used to determine if there is a significant difference between the sample mean/median and a hypothesised mean/median value of 3 (representing neutrality). The conducting of significance tests is guided by a 5-stage framework suggested in Agresti (2018, p. 140). The framework consists of *assumptions*, *hypotheses*, *test statistic*, *p-value* and *conclusions* about the data. A hypothesis testing approach is suggested where the null hypothesis (H_0) is a statement that the test parameter assumes a specific neutral value or a range of values and the alternate hypothesis (H_a) assumes an alternative range of values. In the context of the current study where the data is structured according to Likert scale responses, *H_0 will assume the neutral value of 3*. In the case of the parametric approach, the test statistic that will be used is the mean (H_0: M=3) and in the case of the non-parametric equivalent, the test statistic that will be used is the median (H_0: *Mdn*=3). The alternate hypothesis is that the mean and median are significantly different from the neutral value of 3 (i.e. $H_a \neq 3$). Depending on the assumptions regarding the type of data, the one sampled t-test or the Wilcoxon one-sample signed ranked test will be used to determine if the null hypothesis may be rejected or upheld.

Presentation and Analysis of Perceived Usefulness (PU) of SDOM

The Likert scale responses (6 items) for the construct of PU is illustrated as aggregated percentages in Figure 6.5.

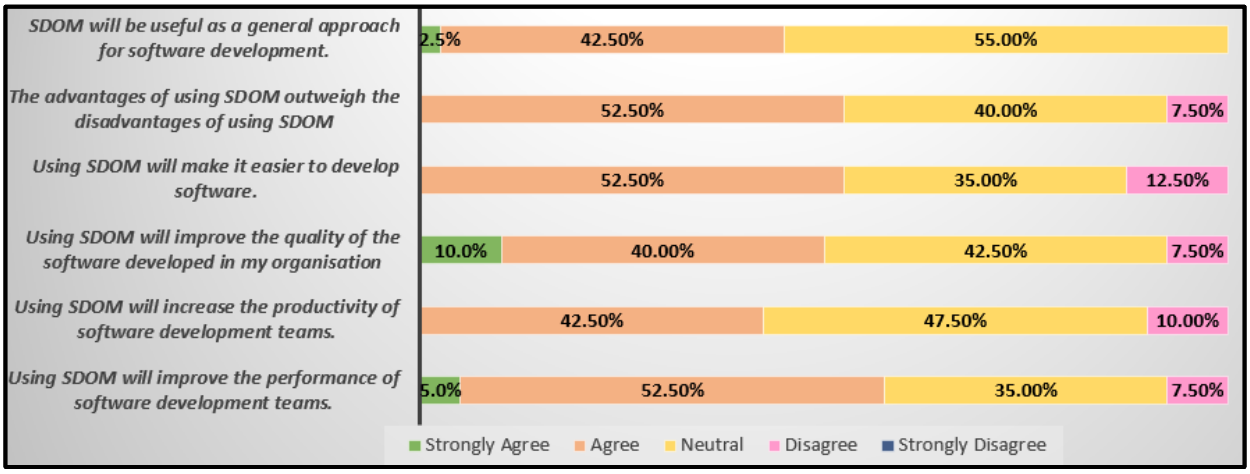


Figure 6.5: Aggregated Percentages for PU

In order to obtain initial overview knowledge of the data presented in Figure 6.5, the mean responses were classified into broader nominal intervals that entailed a conflation of the 5 Likert scale options into 3 categories labelled *negative* (to represent strongly disagree and disagree), *neutral* (to represent neutral) and *positive* (to represent agree and strongly agree). The Likert scale design strategy used for the questionnaire entailed the use of positively phrased 'stem' statements so that the coding approach entailed the allocation of lower values (1 and 2) to a negative response, 3 represented a neutral response and the higher values (4 and 5) represented a positive response. This approach of refining the rating scale so that overview knowledge of the data may be obtained is aligned to the suggestions in Huck (2012, p. 425) and Lovelace and Brickman (2013). A graphical frequency based overview of the responses to the construct of PU using the refined classification is illustrated in Figure 6.6.

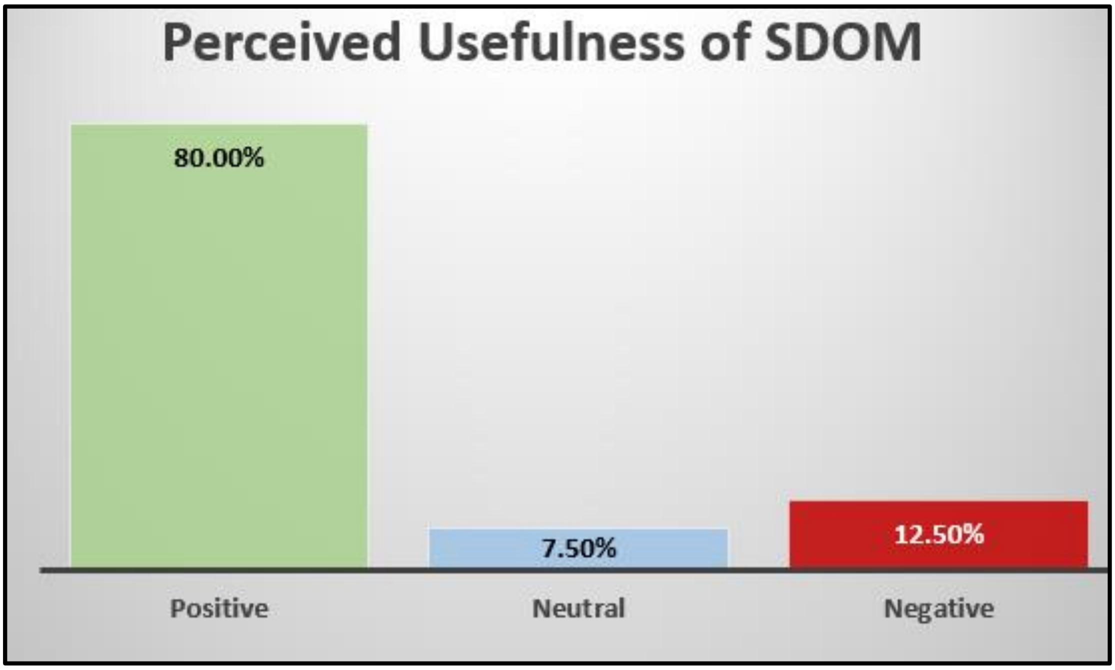


Figure 6.6: Frequency Based Indicator of PU of SDOM

As can be observed in Figure 6.6, the majority (80%, n=32) of the respondents have a positive disposition towards SDOM and perceive the model to be useful to enhance the software development process in their organisation. The reduced/conflated scale also serves a secondary purpose by enabling the computation of a Chi-Square (χ^2) goodness of fit test statistic that provides an indicator as to whether there is a significant difference between the *observed* frequencies from Figure 6.6 and the *expected* frequencies for the categories of positive, neutral and negative. The reduction in the number of categories ensures that there is no possibility of a violation of the basic assumption underling the χ^2 test (i.e. for the given data set, the minimum number of expected values in each category must exceed 5). The χ^2 test uses a null hypothesis that indicates that there is no significant difference between the observed frequencies and the expected frequencies. The results of the χ^2 for the observed frequencies illustrated in Figure 6.6 suggests a rejection of the null hypothesis. The majority preference for the positive option for the PU of SDOM is statistically significant ($\chi^2(2, 40)=39.35$, $p<0.01$).

To determine the significance of the measures of central tendency for PU (6 items), reference is drawn to the original 5 point Likert scale items that are used as the data source for the histogram illustrated in Figure 6.7. Included in Figure 6.7 is a report of the mean (*M*=3.47 and *SD* = 0.461) and median (*Mdn*= 3.5).

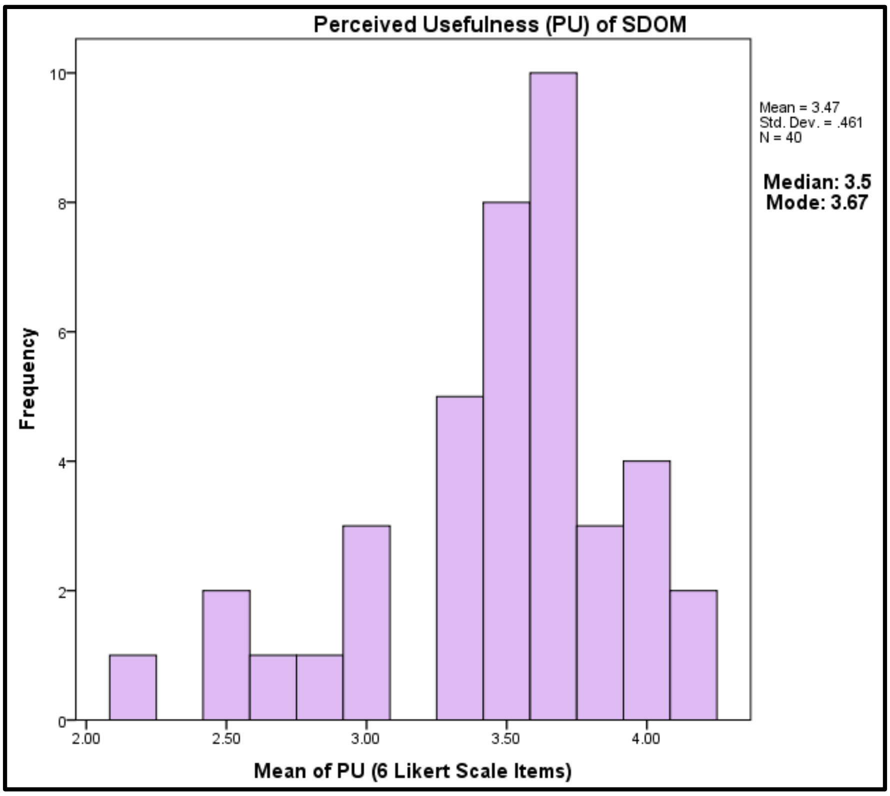
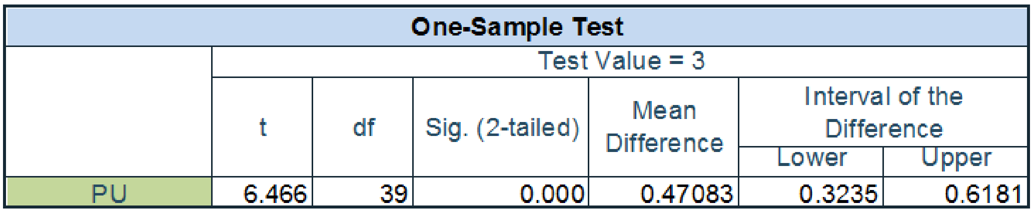


Figure 6.7: Histogram and Central Tendency Data for PU of SDOM

The skewness is reported at -1.014 and the kurtosis is reported at 0.914. According to Kim (2013), if the absolute values of these measures are less than 1.96, then the assumption of normality is upheld. The assumption of normality is somewhat blurred by the result from the SW test that suggests a non-normal distribution for PU. Hence, there is an argument for parametric as well as non-parametric testing that may be used to establish the significance of the measures of central tendency. In the context of the Agresti (2018) five stage framework, the assumptions that can be made is that the sample distribution warrants both a parametric as well as a non-parametric approach. In both instances, $p<0.05$ will be used as the indicator for the acceptance of the null hypothesis.

For the ‘parametric version’ of PU significance test, a one sample t-test was conducted on the significance of the observed mean of 3.5. The results of the one sample t-test are reported in Table 6.5.

Table 6.5: One Sampled t-test for PU



One-Sample Test						
	Test Value = 3					
	t	df	Sig. (2-tailed)	Mean Difference	Interval of the Difference	
					Lower	Upper
PU	6.466	39	0.000	0.47083	0.3235	0.6181

As can be observed in Table 6.5, there is a statistically significant difference between the hypothesised mean and the observed mean at the 95% confidence level ($p<0.05$), suggesting a *rejection of the null hypothesis* and an acceptance of the alternate hypothesis (H_a: $M \neq 3$). In order to determine if the observed mean is significantly *greater* than the hypothesized mean, the null and alternate hypotheses are changed to read H_0: $M \leq 3$ and H_a: $M > 3$ respectively. A one tail-t test is computed to determine if the null hypothesis is rejected or upheld. Although the SPSS package does not provide the results for a one-tailed t-test, the parameters for the 2-tailed t-test shown in Table 6.5 can be adjusted to provide a t value that may be compared with a critical value from the Student’s t distribution table. From this table, it is reported that df(39) = 1.685 which provides a boundary/critical value for the region of rejection of the null hypothesis. From the results of the One-sample t-test ($t(40) = 6.47$, $p<0.01$), the null hypothesis is rejected suggesting that the sample mean is significantly greater than the hypothesised mean of 3.

This result is also confirmed in the non-parametric equivalent tests of significance. The Wilcoxon-one sample signed ranked test of the sample median (*Mdn*=3.5) against the hypothesised median value (*Mdn*=3). The results are illustrated in Figure 6.8.

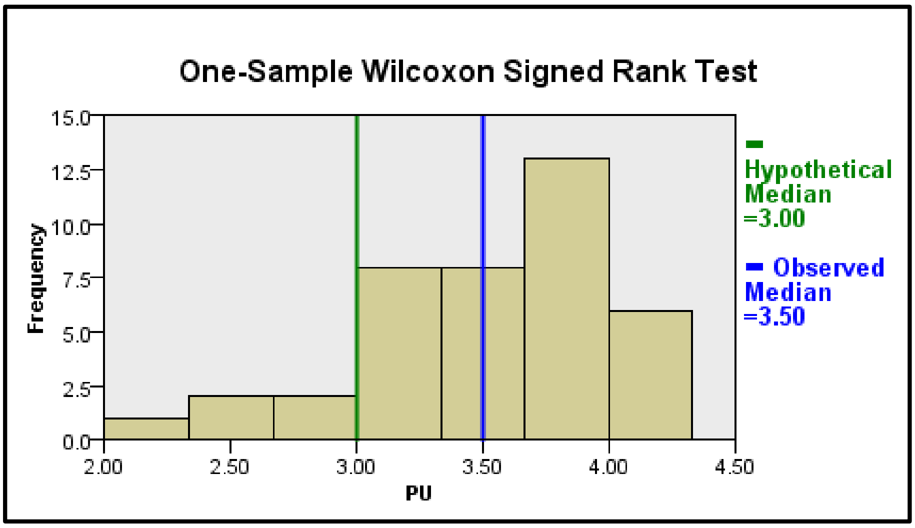


Figure 6.8: Non Parametric test of the Median

As can be seen from Figure 6.8, the observed median is significantly (p<0.05) greater than the hypothesised median of 3.

Presentation and Analysis of the Compatibility (CO) of SDOM

The Likert scale responses (3 items) for the construct of CO is illustrated as aggregated percentages in Figure 6.9.

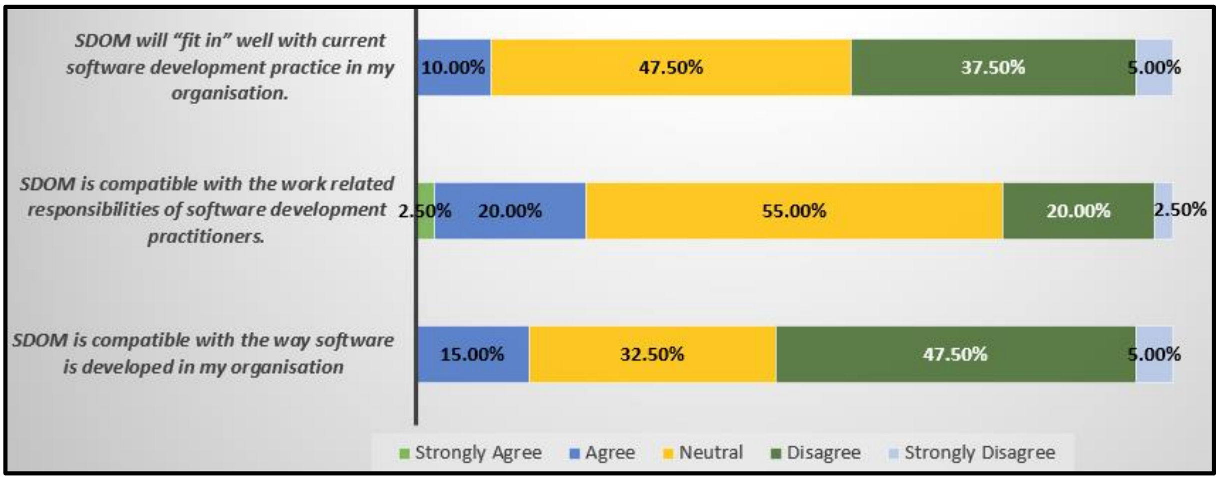


Figure 6.9: Aggregated Percentages for Compatibility of SDOM

An exercise in conflating the 5 Likert scale categories into 3 categories was once more undertaken (similar to PU) so that an initial overview understanding of the data pattern could be obtained for the Compatibility construct. The outcome of this exercise is illustrated graphically in Figure 6.10.

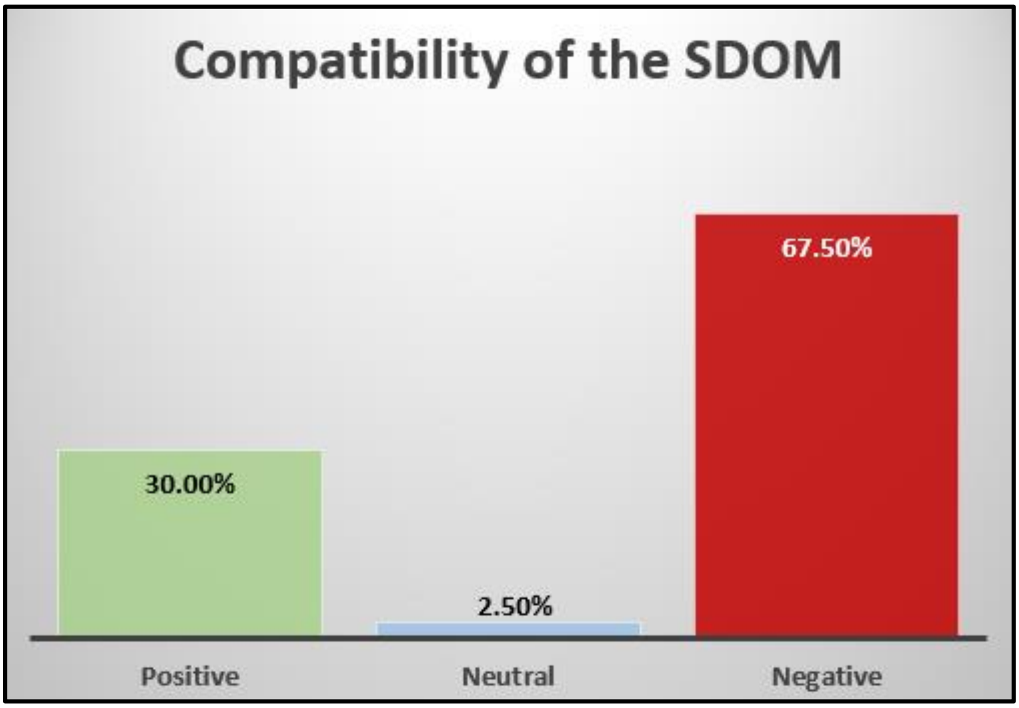


Figure 6.10: Frequency Based Indicator of Compatibility of SDOM

As can be observed in Figure 6.10, the majority (67.5%, n=27) of the respondents have a negative disposition towards SDOM and perceive the model to be incompatible with their current software development process. The results of the χ^2 test for the observed frequencies illustrated in Figure 6.10 suggests a rejection of the null hypothesis that the observed frequencies are equal to the expected frequencies ($\chi^2(2, 40)=25.5$, $p<0.01$). The majority of the responses on the compatibility of SDOM to current organisational software development processes is negative.

To determine the significance of the measures of central tendency for CO (3 items), reference is drawn to the original 5 point Likert scale items that are used as the data source for the histogram illustrated in Figure 6.11. Included in Figure 6.11 is a report of the observed mean value (*M*=2.7) and the observed median (*Mdn*=2.7). It should be noted that the skewness is reported at -0.032 and the kurtosis is reported at -0.099. The absolute values for both these measures are less than 1.96 thus suggesting that the assumption of normality may be upheld. The assumption of normality is somewhat blurred by the result from the SW test that suggests a non-normal distribution for the Compatibility construct. Based on these deliberations, there is an argument for parametric as well as non-parametric testing. The measures of central tendency will be subjected to parametric tests and there will be an attempt to corroborate these results with the non-parametric version. An illustrative view of the sample distribution for CO is presented as a histogram illustrated in Figure 6.11.

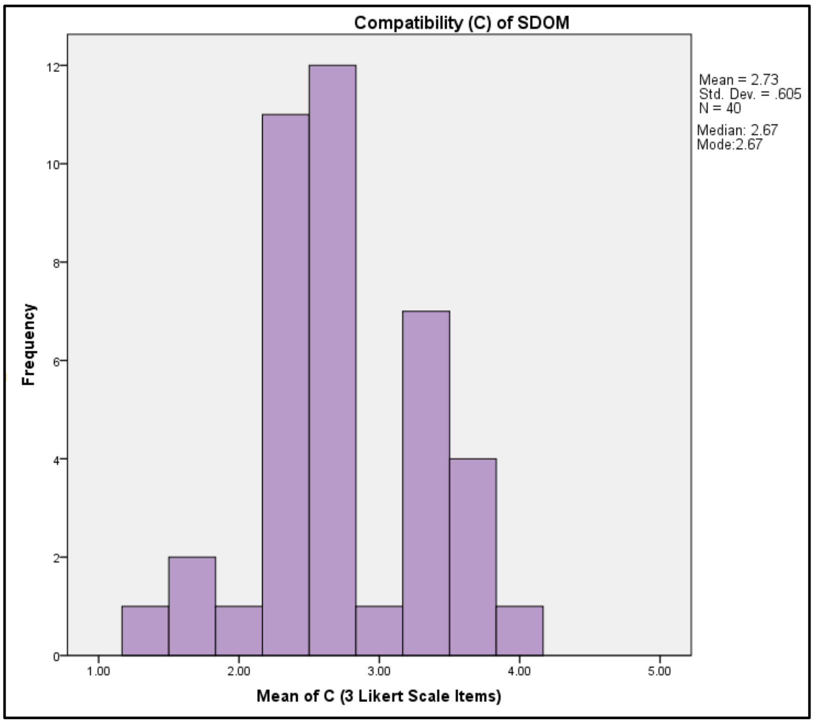
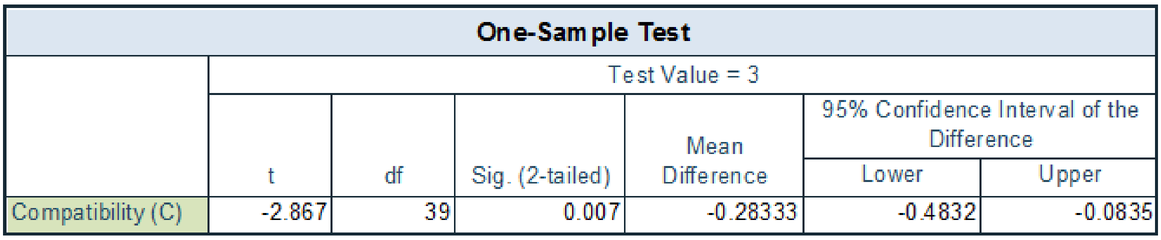


Figure 6.11: Histogram and Central Tendency Data for CO of SDOM

To determine if the observed mean (*M*=2.7) is significantly different from the hypothesised neutral value of 3 (H_0: *M*=3), a one sample t-test was conducted. The result of the t-test is reported in Table 6.6.

Table 6.6: One Sampled t-test for Compatibility of SDOM



One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Compatibility (C)	-2.867	39	0.007	-0.28333	-0.4832	-0.0835

As can be observed in Table 6.6, it can be concluded that the observed mean is significantly different from the hypothesised mean (p<0.05), thereby suggesting a rejection of the null hypothesis and an acceptance of the alternate hypothesis (H_a: $M \neq 3$). In order to determine if the observed mean is significantly *less* than the hypothesized mean, the null and alternate hypotheses are changed to read H_0: $M >= 3$ and H_a: $M<3$ respectively. A one tail-t test is computed to determine if the null hypothesis is rejected or upheld. A comparison of the t value from Table 6.6 (t=-2.867) with the critical value from the Student's t distribution table (df(39) =1.685) indicates that the observed t statistic (t(40)=-2.867, p<0.05) lies to the left of the boundary/critical value of 1.685. The observed mean is thus significantly less than the hypothesised mean suggesting a rejection of the null hypothesis.

As a confirmatory exercise, the Wilcoxon-one sample signed ranked test of the sample median value was tested against a hypothesised median value of 3. The results are illustrated in Figure 6.12.

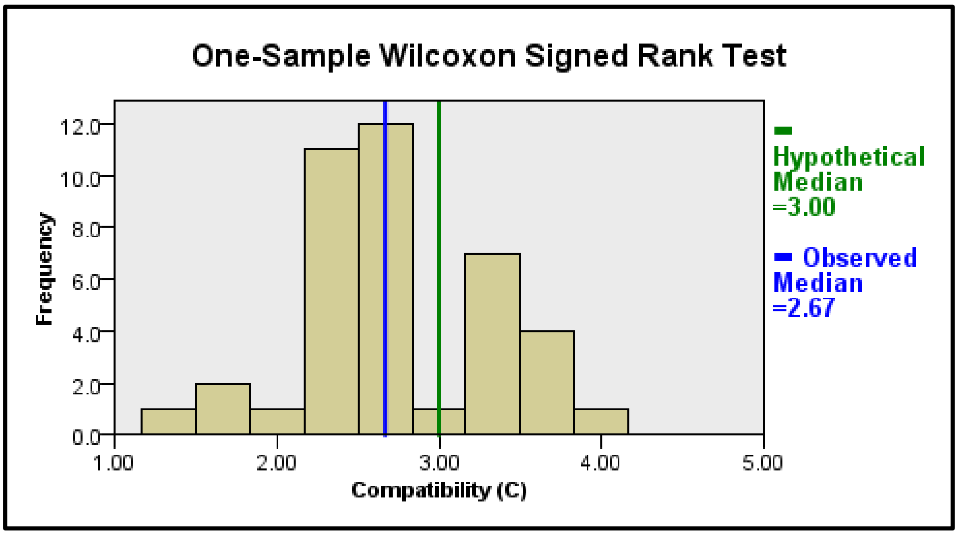


Figure 6.12: Non Parametric test of the Median Value for Compatibility

As can be seen from Figure 6.12, the observed median (*Mdn*=2.67) is significantly (p<0.05) *less* than the hypothesised median (*Mdn*=3).

Presentation and Analysis of the Subjective Norm (SN) of SDOM

The Likert scale responses (3 items) for the construct of SN is illustrated as aggregated percentages in Figure 6.13.

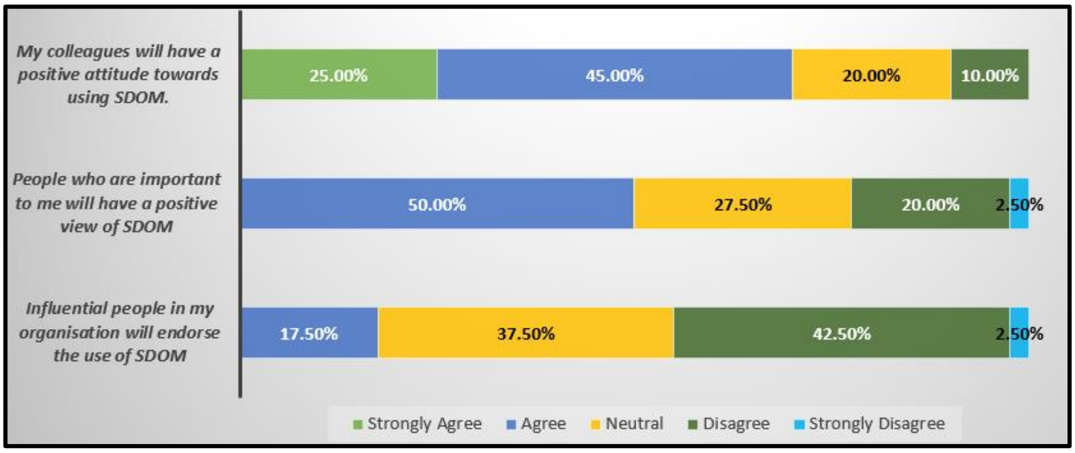


Figure 6.13: Aggregated Percentages for the Subjective Norm of SDOM

An exercise in conflating the 5 Likert scale categories into 3 categories was undertaken for the construct of SN so that an initial overview understanding of the data pattern for SN could be obtained. The outcome of this exercise is illustrated graphically in Figure 6.14

As can be observed in Figure 6.14, the majority (60%, n=24) of the respondents have a positive disposition towards the SN of using SDOM. The results of the χ^2 test for the observed frequencies illustrated in Figure 6.14 suggest a rejection of the null hypothesis that the observed frequencies are equal to the expected frequencies ($\chi^2(2, 40)=7.5$, $p<0.05$). The majority of the responses with regards to the SN of using SDOM in an organisational context is significantly positive.

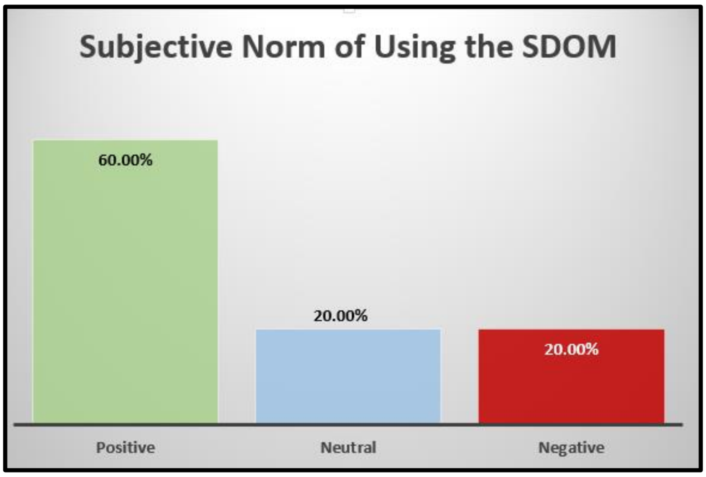


Figure 6.14: Frequency Based Indicator of the SN of Using SDOM

To determine the significance of the measures of central tendency for SN (3 items), reference is drawn to the original 5 point Likert scale items that are used as the data source for the histogram illustrated in Figure 6.15. Included in Figure 6.15 is a report of the mean value (*M*=3.24, *SD*=.613) and the median (*Mdn*=3.33). In the case of the SN, the SW test as well as the skewness and kurtosis tests reveal that the sample distribution is normal. Hence the exclusive reliance on the parametric testing option is warranted.

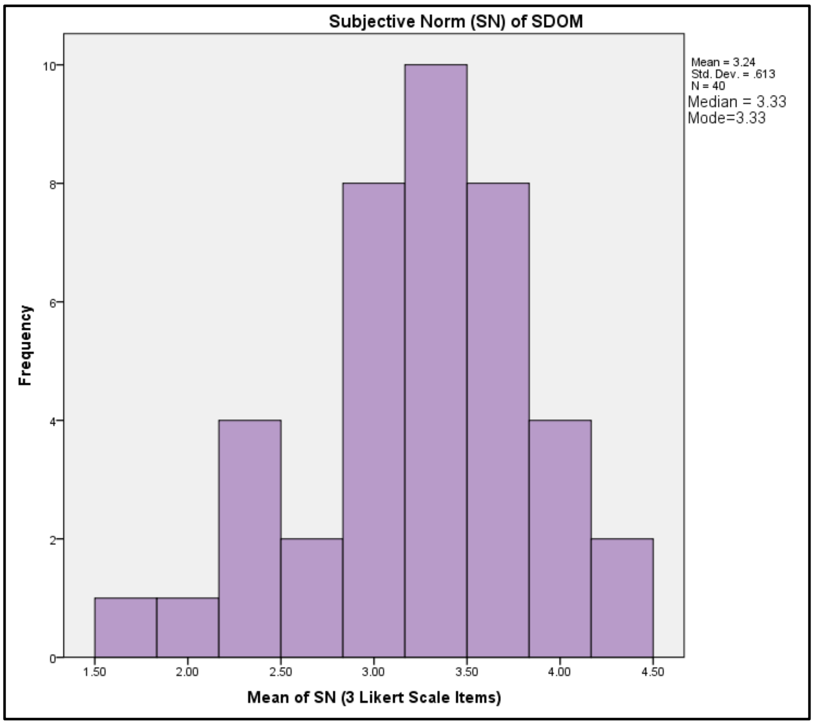
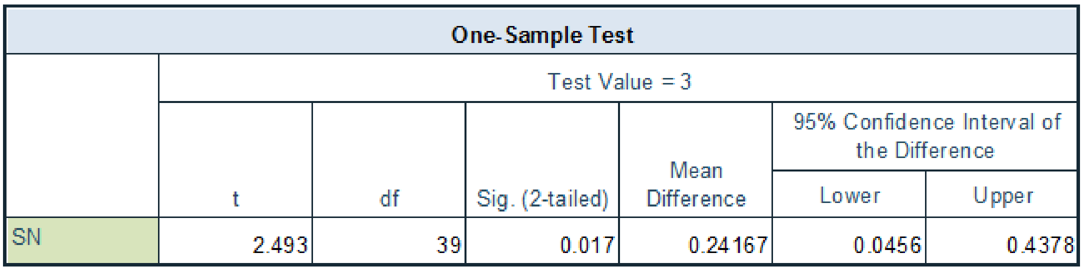


Figure 6.15: Histogram and Central Tendency Data for SN of SDOM

The one sample t-test was conducted on the significance of the observed mean (*M*=3.24). The results of the one sample t-test are reported in Table 6.7.

Table 6.7: One Sampled t-test for Compatibility of SDOM



One-Sample Test						
	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SN	2.493	39	0.017	0.24167	0.0456	0.4378

As can be observed in Table 6.7, there is a statistically significant difference between the hypothesised mean and the observed mean for SN at the 95% confidence level (p<0.05), suggesting a rejection of the null hypothesis and an acceptance of the alternate hypothesis (H_a: $M \neq 3$). In order to determine if the observed mean for SN is significantly *greater* than the hypothesized mean, the null and alternate hypotheses are changed to read H_0:$M \leq 3$ and H_a: $M>3$ respectively. A one tail-t test is computed to determine if the null hypothesis is rejected or upheld. The Student's t distribution table, df(39) = 1.685 is used to obtain a boundary/critical value for the region of rejection of the null hypothesis. The results of the t test show that the t statistic (t(40)=2.49, p<0.05) lies to the right of the boundary value thereby enabling a rejection of the null hypothesis and paving the way for the conclusion that the sample mean is significantly *greater* than the hypothesised mean of 3.

Presentation and Analysis of the Organisational Support (OS) for SDOM

The Likert scale responses (2 items) for the construct of OS is illustrated as aggregated percentages in Figure 6.16.

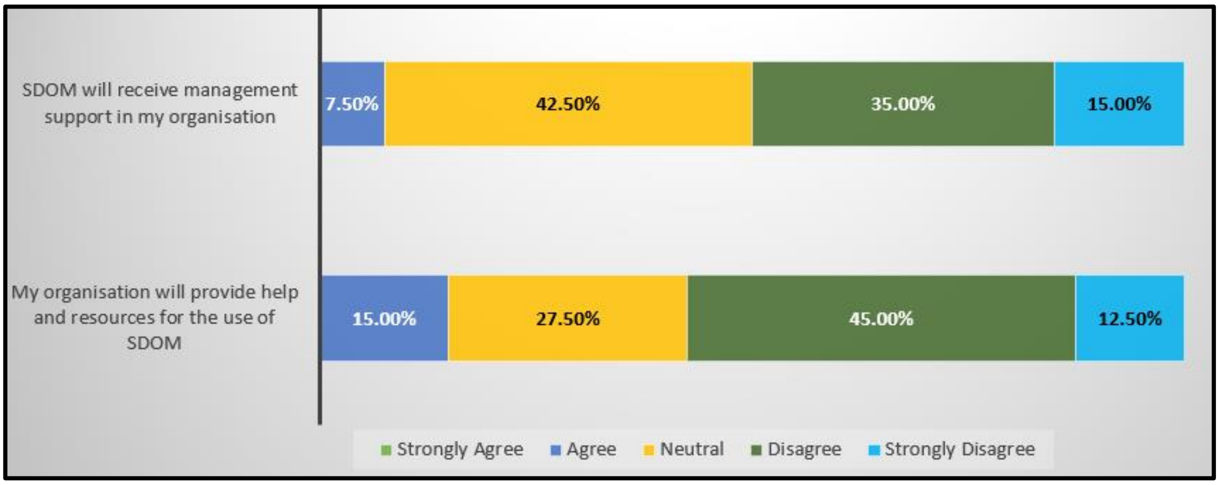


Figure 6.16: Aggregated Percentages for the OS for the SDOM

An exercise in conflating the 5 Likert scale categories into 3 categories was undertaken for the construct of OS so that an initial overview understanding of the data pattern for OS could be obtained. The outcome of this exercise is illustrated graphically in Figure 6.17.

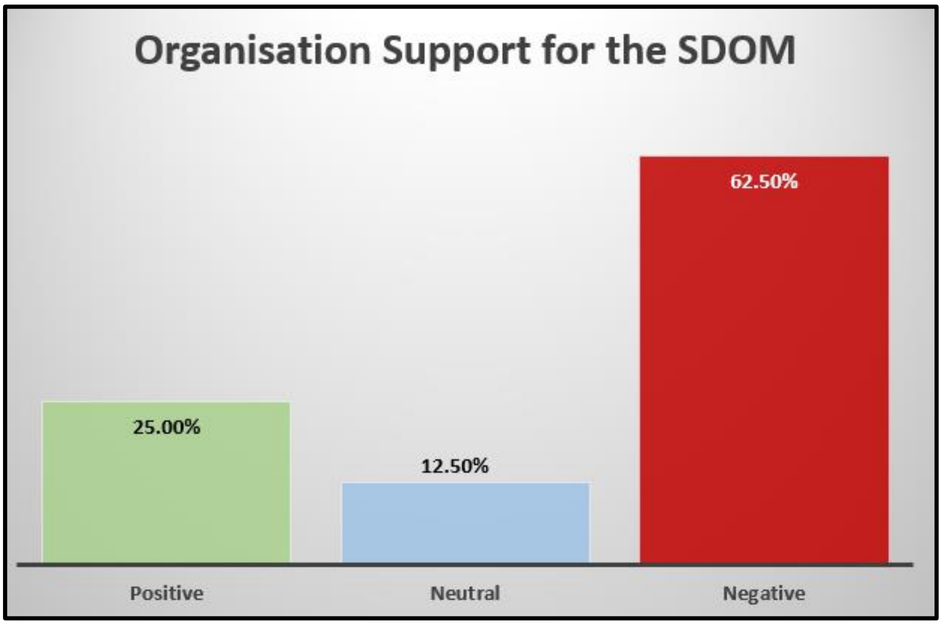


Figure 6.17: Frequency Based Indicator of the OS for Using SDOM

As can be observed in Figure 6.17, the majority (62.5%, n=25) of the respondents have a negative perception of the organisational support that SDOM will receive. The results of the χ^2 test for the observed frequencies illustrated in Figure 6.17 suggest a rejection of the null hypothesis that the observed frequencies are equal to the expected frequencies ($\chi^2(2, 40)=16.25$, $p<0.01$). The majority of the responses with regards to the OS for using SDOM in an organisational context is significantly negative.

To determine the significance of the measures of central tendency for OS (2 items), reference is drawn to the original 5 point Likert scale items that are used as the data source for the histogram illustrated in Figure 6.18. Included in Figure 6.18 is a report of the mean (*M*=2.6, *SD*=0.78) and median (*Mdn*=2.5). It should be noted that the SW test for normality indicates a non-normal distribution. However, the tests for skewness and kurtosis falls within the range of acceptability indicating that the violation of the assumption of normality is not severe enough to eliminate the prospect of parametric testing. However, in order to avoid any element of doubt, the significance of the mean and median values for OS will be subjected to parametric and non-parametric tests.

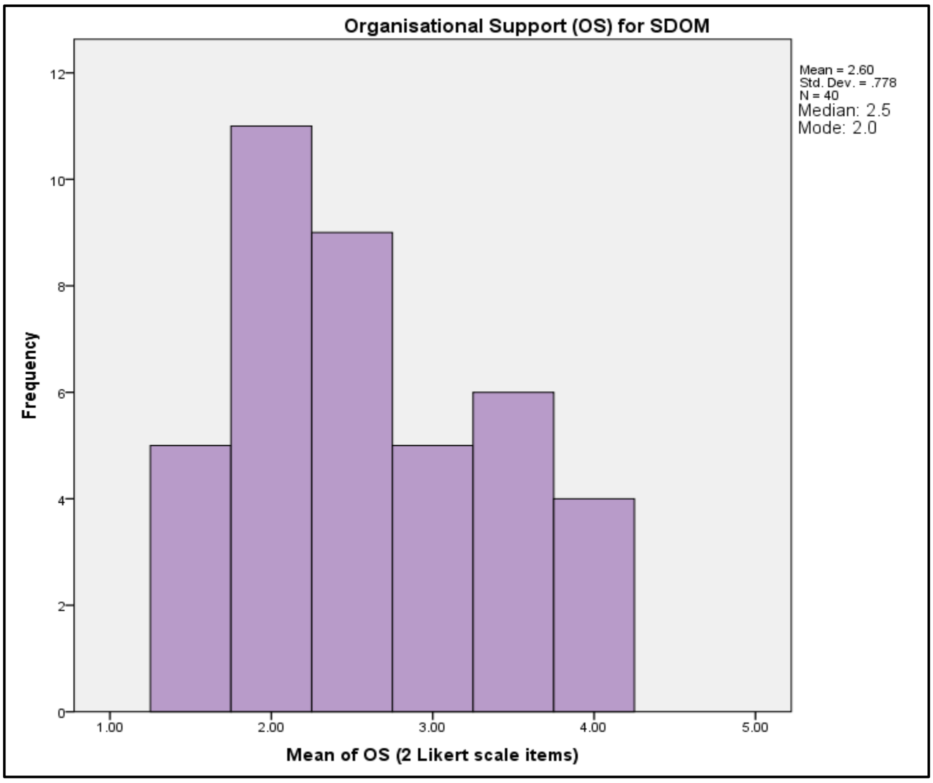
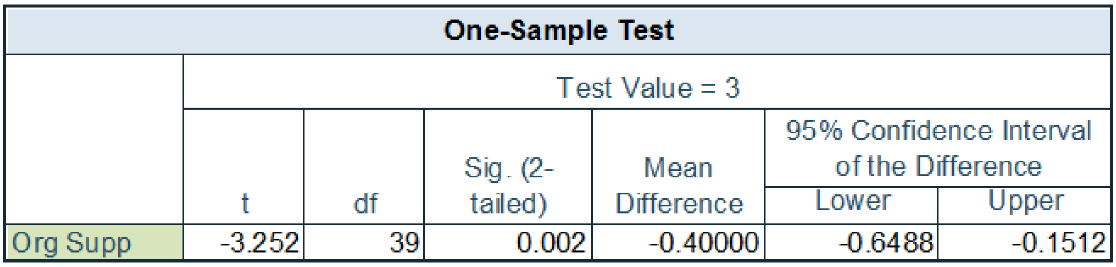


Figure 6.18: Histogram and Central Tendency Data for OS for SDOM

The one sample t-test was conducted on the significance of the observed mean of 2.60. The results of the one sample t-test are reported in Table 6.8.

Table 6.8: One Sampled t-test for OS for SDOM



One-Sample Test						
	Test Value = 3					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
Org Supp	-3.252	39	0.002	-0.40000	-0.6488	-0.1512

As can be observed in Table 6.8, it can be concluded that the observed mean for OS is significantly different from the hypothesised mean (p<0.05), thereby suggesting a rejection of the null hypothesis and an acceptance of the alternate hypothesis (H_a: $M \neq 3$)

In order to determine if the observed mean is significantly *less* than the hypothesized mean, the null and alternate hypotheses are changed to read H_0: M

≥3 and H_a:M<3 respectively. A one tail-t test is computed to determine if the null hypothesis is rejected or upheld. A comparison of the t value from Table 6.8 (t=-3.252) with the critical value from the Student's t distribution table (df(39) =1.685) indicates that the observed statistic (t(40)=-3.252, p<0.05) lies to the left of the boundary/critical value of 1.685. The observed mean is thus significantly less than the hypothesised mean suggesting a rejection of the null hypothesis and acceptance of the alternate hypothesis that the mean for OS is *significantly less* than the hypothesised mean value of 3.

As a confirmatory exercise, the Wilcoxon-one sample signed ranked test of the sample median value was tested against a hypothesised median value of 3. The results are illustrated in Figure 6.19.

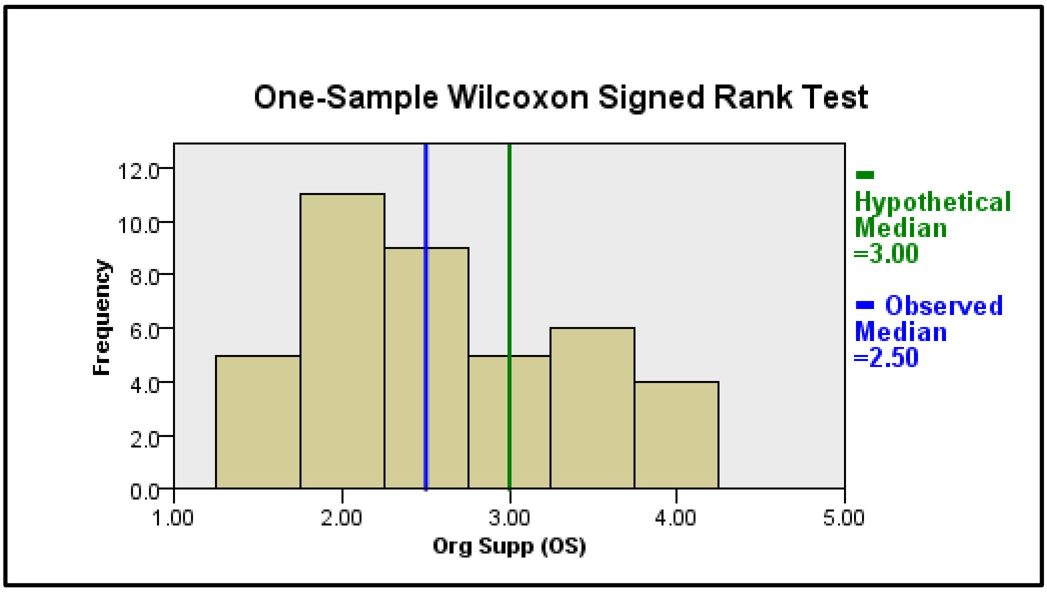


Figure 6.19: Non Parametric test for Organisational Support

As can be seen from Figure 6.19, the non-parametric tests of significance indicate that the observed median is significantly (p<0.05) *less* than the hypothesised median of 3. This corroborates the outcome of the equivalent parametric test.

Presentation and Analysis of the Behavioural Intention (BI) to use SDOM

The Likert scale responses (2 items) for the construct of BI is illustrated as aggregated percentages in Figure 6.20

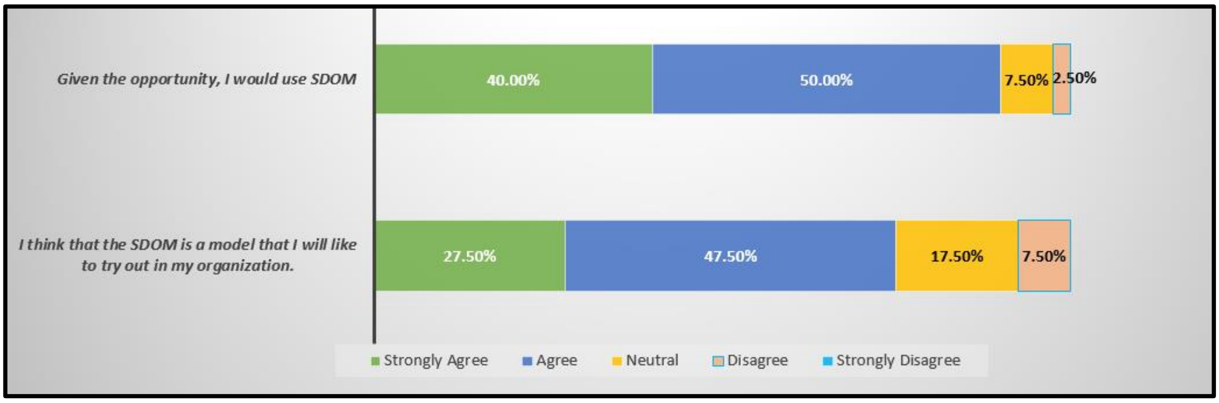


Figure 6.20: Aggregated Percentages for the BI to Use SDOM

An exercise in conflating the 5 Likert scale categories into 3 categories was undertaken for the construct of BI so that an initial overview understanding of the data pattern for BI could be obtained. The outcome of this exercise is illustrated graphically in Figure 6.21

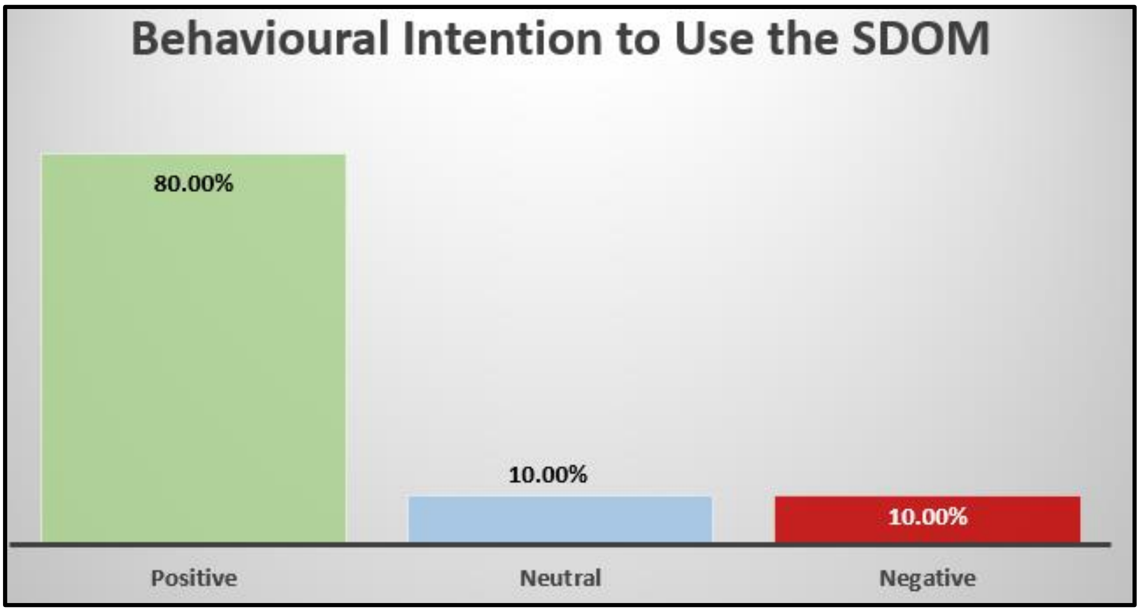


Figure 6.21: Frequency Based Indicator of the BI to use SDOM

As can be observed in Figure 6.21, the majority (80%, n=32) of the respondents have a positive disposition towards an intention to use SDOM in an organisational context for software development projects. The results of the χ^2 test for the observed frequencies illustrated in Figure 6.21 suggest a rejection of the null hypothesis that the observed frequencies are equal to the expected frequencies (χ^2(2, 40)=39.2, p<0.01). The majority of the responses with regards to a behavioural intention to make use of SDOM if an opportunity arises within an organisation is significantly positive.

To determine the significance of the measures of central tendency for BI (2 items), reference is drawn to the original 5 point Likert scale items that are used as the data source for the histogram illustrated in Figure 6.22. Included in Figure 6.22 is a report of the mean (*M*=3.84, *SD*= 0.77) and the median (*Mdn*=4). It should be noted that the SW test for normality indicates a non-normal distribution. However, the tests for skewness and kurtosis falls within the range of acceptability indicating that the violation of the assumption of normality is not severe enough to eliminate the prospect of parametric testing. However, in order to avoid any element of doubt, the significance of the mean and median values for BI will be subjected to parametric and non-parametric tests.

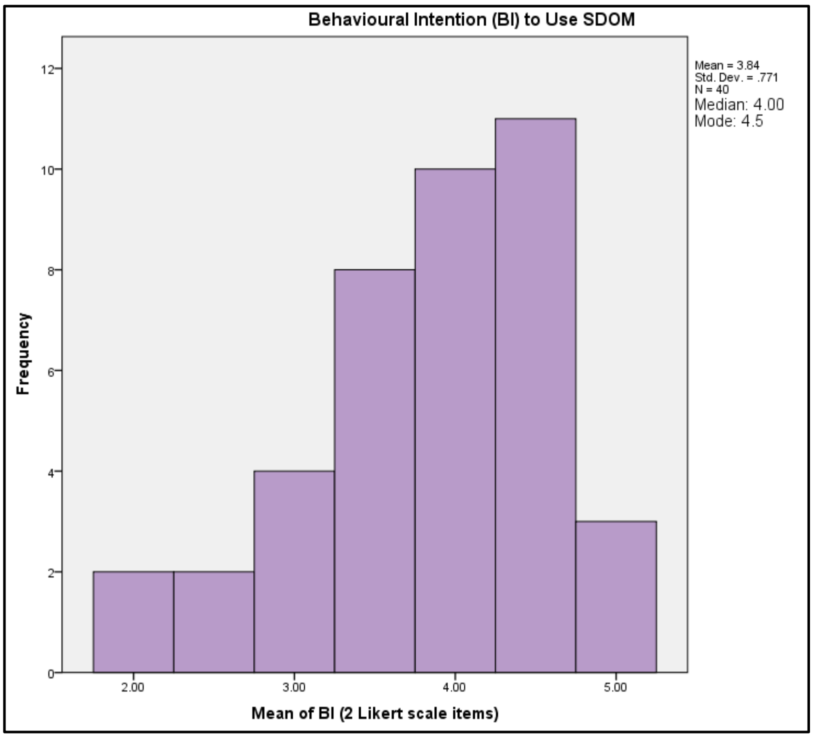
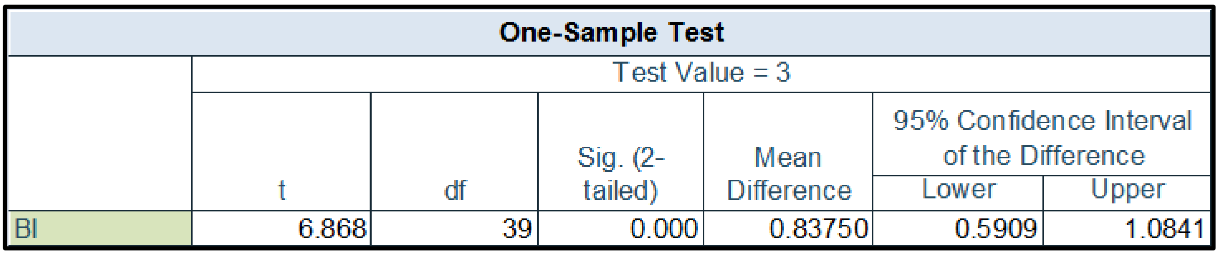


Figure 6.22: Histogram and Central Tendency Data for BI for SDOM

The one sample t-test was conducted on the significance of the observed mean of 3.84. The results of the one sample t-test are reported in Table 6.9.

Table 6.9: One Sampled t-test for OS for SDOM



One-Sample Test						
	Test Value = 3					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
BI	6.868	39	0.000	0.83750	0.5909	1.0841

As can be observed Table 6.9, it can be concluded that the observed mean for BI is significantly different from the hypothesised mean (p<0.05), thereby suggesting a rejection of the null hypothesis and an acceptance of the alternate hypothesis (H_a: *M* #3)

In order to determine if the observed mean is significantly *greater* than the hypothesized mean, the null and alternate hypotheses are changed to read H_0: *M* ≤3 and H_a: *M* >3 respectively. A one tail-t test is computed to determine if the null hypothesis is rejected or upheld. A comparison of the t value from Table 6.9 (t=6.868) with the critical value from the Student's t distribution table (df(39) =1.685) indicates that the observed statistic (*t*(40)=6.868, *p*<0.05) lies to the right of the boundary/critical value of 1.685. The observed mean is thus significantly greater than the hypothesised mean suggesting a rejection of the null hypothesis and acceptance of the alternate hypothesis that the mean for BI is *significantly greater* than the hypothesised mean value of 3.

As a confirmatory exercise, the Wilcoxon-one sample signed ranked test of the sample median value was tested against a hypothesised median value of 3. The results are illustrated in Figure 6.23. As can be seen from Figure 6.23, the non-parametric tests of significance indicate that the observed median is significantly (p<0.05) *greater* than the hypothesised median of 3 for the construct of BI.

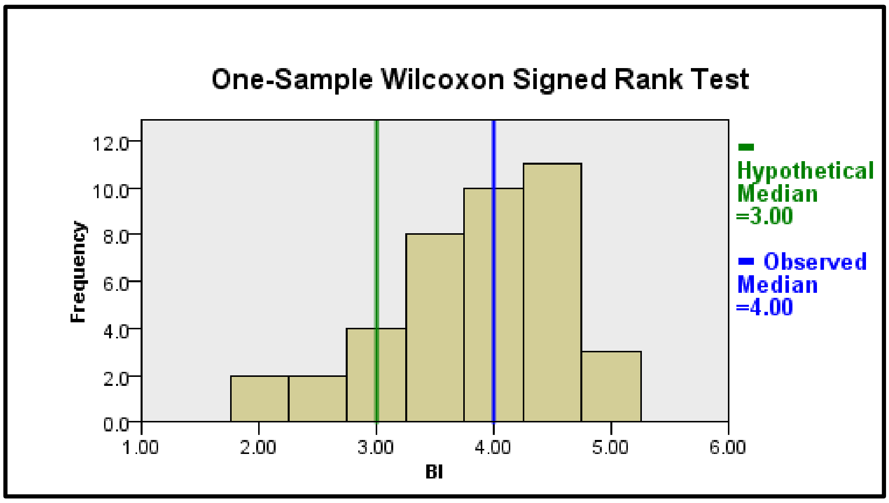


Figure 6.23: Non Parametric test for the Construct of BI

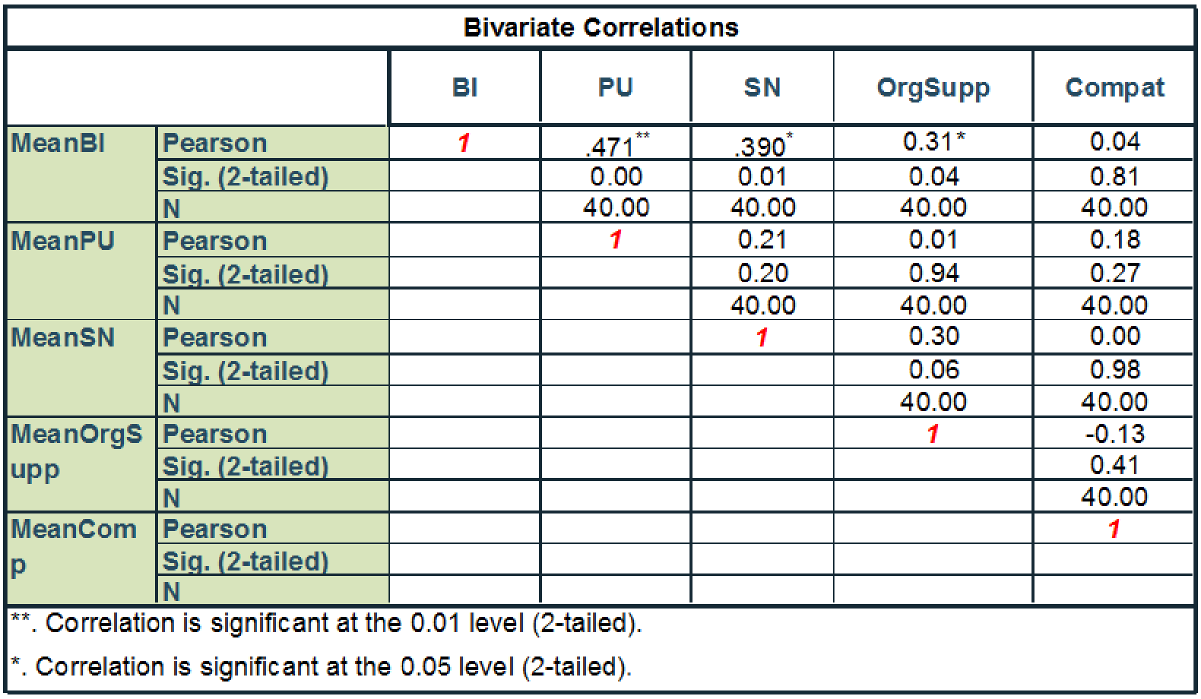
The result from Figure 6.23 corroborates the outcome of the equivalent parametric test suggesting that the measure(s) of centrality for BI is significantly greater than the neutral value of 3.

Bivariate Analysis of the Constructs from SDOM

Correlation analysis is a bivariate statistical technique that describes the relationship(s) between the variables of a study. However, as Gravetter and Wallnau (2014) point out, correlation analysis does not represent a cause-and-effect relationship between the variables of a study and should not be used to generalise the correlation beyond the range of the data represented in the sample unless there is a wide range of data values to work with. As an affirmation of these principles in the context of the current study, the correlation analysis undertaken is used to describe the relationship(s) between the constructs (PU, CO, SN, OS and BI) from the TASDM theoretical model that underpin the quantitative phase of the study. As a disclaimer, there is no intention to propose a cause-and-effect relationship between the variables or to extrapolate these relationship(s) to a domain beyond the confines of the data gathered as part of the current study. According to Gravetter and Wallnau (p. 450), a correlation is a numerical value that provides an indicator of 3 attributes of a relationship between variables in a study. These are the direction (positive or negative), form (linear or non-linear) and strength (value ranging from -1 to +1) of the relationship. The most common correlation standard used is the Pearson product-moment correlation which measures the degree of the straight-line relationship between variables in a study. The direction and strength of this relationship is represented by the value attached to the letter r, also referred to as the sample correlation co-efficient. Attached to the r value is a confidence level statistic (p value) that provides an indicator of the robustness of the correlation value if more data points were added to the sample set.

A Pearson product-moment correlation analysis was conducted to examine the relationship between PU, CO, SN, OS and BI. The outcome of this correlation analysis is illustrated as a correlation matrix in Table 6.10. As can be seen in Table 6.10, the relationship between Perceived Usefulness (PU) of SDOM and Behavioural Intention (BI) is a significant positive correlation (r(38)=0.47, p<0.01). Subjective Norm (SN) also has a significant positive relationship with BI (r (38) =0.39, p<.0.05) and so does organisational support (r 38) =0.31, p<0.05). However, the relationship between the Compatibility (CO) of SDOM and BI was not a significant one (r (38) =0.04, p>0.05).

Table 6.10: Pearson product-moment correlation analysis of PU, CO, SN, OS and BI



Bivariate Correlations

		BI	PU	SN	OrgSupp	Compat
MeanBI	Pearson	1	.471**	.390*	0.31*	0.04
	Sig. (2-tailed)		0.00	0.01	0.04	0.81
	N		40.00	40.00	40.00	40.00
MeanPU	Pearson		1	0.21	0.01	0.18
	Sig. (2-tailed)			0.20	0.94	0.27
	N			40.00	40.00	40.00
MeanSN	Pearson			1	0.30	0.00
	Sig. (2-tailed)				0.06	0.98
	N				40.00	40.00
MeanOrgSupp	Pearson				1	-0.13
	Sig. (2-tailed)					0.41
	N					40.00
MeanComp	Pearson					1
	Sig. (2-tailed)					
	N					

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

The Pearson correlation computation provides an indication of the bivariate relationships between the main constructs of TASDM. In this regard the main focal point of analysis is the significant relationships between the main constructs of the study. From the bivariate perspective, the only significant relationships are between BI and PU, BI and SN and BI and OS. However, as Remler and Van Ryzin (2011, p. 293) point out, "…the real world is more than two dimensional-many factors exert their influence at the same time and in complex ways". This comment was made in the context that multiple regression techniques are required to analyse phenomena that are linked to more than a single independent variable.

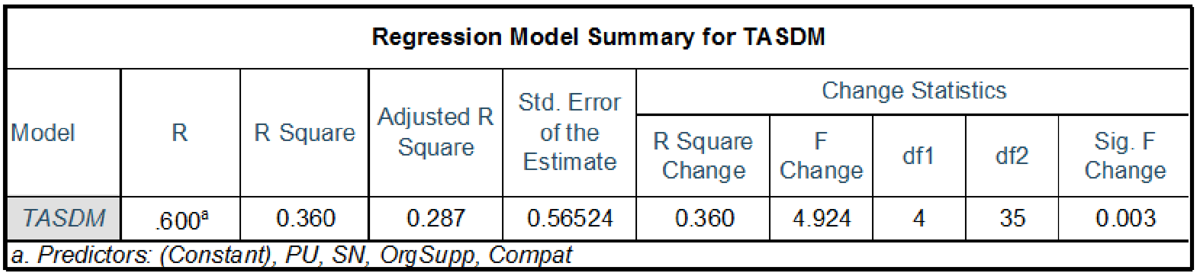
Multivariate Analysis of the Constructs from SDOM

In the context of TASDM, the phenomenon of BI is linked to 4 predictor/independent variables thus necessitating a multiple regression analysis. According to Remler and Van Ryzin, multiple regression is used to predict a dependent variable by integrating multiple independent variables into a multiple regression model.

Based on this assertion, it becomes quite clear that multivariate analysis represents a foray into causal modelling which is not the intention in the context of the current study. However, the objective of multivariate analysis in the current study's context is to obtain an insight into whether the pattern of correlations between the independent variables (IVs) and the dependent variable (DV) 'fits' the pattern predicted by the underlying TASDM. Multivariate analysis also provides knowledge of the inter-correlations between the IV's of the study.

A multiple regression model was developed for the variables in the current study. The BI construct was specified as the DV and PU, CO, SN and OS were specified as the IVs. The overall multiple regression model with all 4 IVs produced $R^2 = .36$, F (4, 35) =4.92, $p<0.01$. This outcome is illustrated in Table 6.11.

Table 6.11: Model Summary for Multiple Regression Analysis of TASDM



Regression Model Summary for TASDM

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
TASDM	.600[a]	0.360	0.287	0.56524	0.360	4.924	4	35	0.003

a. Predictors: (Constant), PU, SN, OrgSupp, Compat

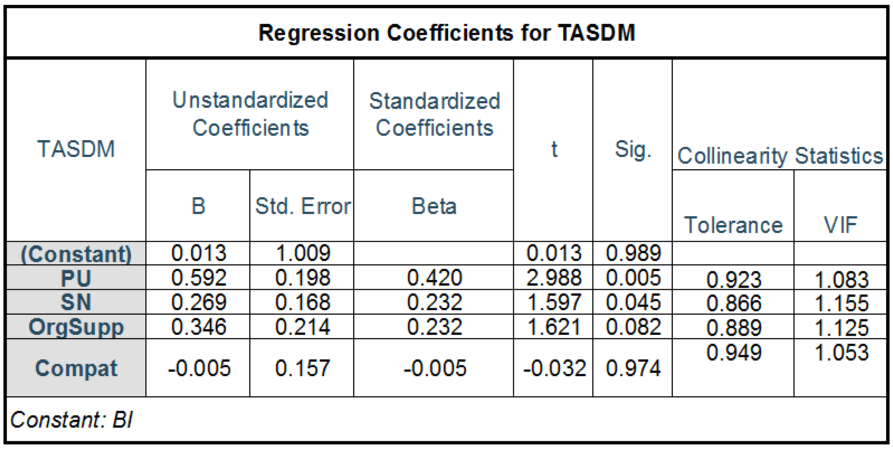
The model summary illustration provides evidence to answer the question:

If the constructs OS, PU, CO, and SN are evaluated as a group, do they predict the behavioural intention to use SDOM?

The significance value (p<0.01) provides an indicator that the overall regression model is valid and the amount of variance that can be accounted for in the DV (BI) is 0.36 (36%). Hence, taken as a set, the 4 IVs (main constructs) of TASDM is a significantly reliable predictor of the BI to use SDOM. Remler and Van Ryzin (2011, p. 296) does warn however that the predictive power (robustness) is increased if the adjusted R-squared value is used instead of the R-square value from the model. Using this value, the predictive capacity of the 4 IVs is set at 28.7%.

The next model that has been output from the regression modelling exercise is the set of co-efficient values that examine the influence of the IVs individually. The coefficients model is illustrated in Table 6.12.

Table 6.12: Coefficients Model for Multiple Regression Analysis of TASDM



Regression Coefficients for TASDM

TASDM	Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
	B	Std. Error	Beta			Tolerance	VIF
(Constant)	0.013	1.009		0.013	0.989		
PU	0.592	0.198	0.420	2.988	0.005	0.923	1.083
SN	0.269	0.168	0.232	1.597	0.045	0.866	1.155
OrgSupp	0.346	0.214	0.232	1.621	0.082	0.889	1.125
Compat	-0.005	0.157	-0.005	-0.032	0.974	0.949	1.053

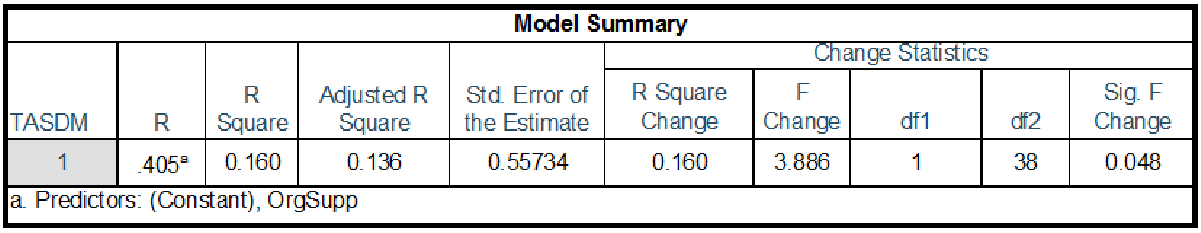
Constant: BI

The first point of contention is the issue of collinearity. When 2 or more variables are highly correlated, then for statistical purposes they are essentially the same variable and both variables cannot be used as independent variables in the multiple regression model (Remler & Van Ryzin, 2011, p. 297). As can be observed in Table 6.12 the variance inflation factors (VIF) are all below 4 indicating that the IVs are not highly correlated with each other thereby suggesting that each of the variables make a unique individual contribution to the overall predictive power of the model.

The Standardised Coefficients values column in Table 6.12 provides an indication of the strength of the unique individual predictive capacity of each of the IVs. In the case of Perceived Usefulness (PU) it is observed that the beta value is 0.42 ($p<0.01$) suggesting that PU uniquely accounts for 42% of the variance in the DV (BI). In the case of Subjective Norm (SN), it is observed that the beta value is 0.23 ($p<0.05$) suggesting that SN uniquely accounts for 23% of the variance in the DV. The amount of variance in the dependent variable that may be attributed to Compatibility (CO) and Organisational Support (OS) is recorded as not significant (in both cases $p>0.05$), suggesting that both these variable do not make a significantly unique contribution in a multivariate context to the predictive capacity of the model. However, at the 90% confidence level ($p<0.1$), OS uniquely accounts for 23% of the variance in the DV.

In order to determine whether the insignificant IVs have a relationship with the significant IVs, a stepwise regression analysis was conducted firstly with PU as the dependent variable and then with SN as the dependent variable. The results of the stepwise regression analysis exercise show that the only significant relationship is between SN and OS. As displayed in Table 6.13, OS is a weak (adjusted R^2 =13.6%) but significant predictor of SN.

Table 6.13: Stepwise Regression with SN as the DV and OrgSupp as the IV



Model Summary

					Change Statistics				
TASDM	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	.405[a]	0.160	0.136	0.55734	0.160	3.886	1	38	0.048

a. Predictors: (Constant), OrgSupp

The results from the regression analysis attest to the finding that PU makes the greatest unique contribution towards explaining the variance in the DV. This is followed by SN. The results for OS are inconclusive at the 95% confidence level in terms of the unique contribution that it makes towards explaining the variance of the DV. An emphatic outcome from the regression analysis is that the construct of compatibility does not make a unique contribution to the overall predictive capacity of the model. These findings will be used in the subsequent sections that entail a Sequential Equation Modelling (SEM) intervention that has an explicit purpose of proposing a model that provides a better predictive capacity of TASDM so that the proposed model will provide a better account of the variability in the DV.

6.4.5 A Structural Equation Modelling (SEM) Intervention

According to Foster *et al.* (2005) SEM is an equation based modelling exercise that examines the relationship between the variables (observed and latent) of a study with the objective of selecting a model that "best fits" the study's data. SEM is regarded as a causal modelling strategy that incorporates methods such as factor analysis, path analysis and correlation-based modelling that represents the researcher's conceptualisation of the study's variables based on the study's data patterns.

In the context of the current study's data patterns, SEM will be ideal to explore some of the anomalies that have thus far been observed between the study's data and the theoretical model. Two significant aspects that necessitate further analysis is the lack of alignment between the Compatibility construct with the theoretical model and the alignment between the constructs of Social Norm and Organisational Support. The initial foray into SEM has been undertaken by virtue of the CFA exercise. The next form of SEM analysis will involve the use of a path analysis diagram that illustrates the relationships between the main constructs and also outputs the predictive capacity of the model by providing an indicator of the amount of variance that the main constructs of the study is able to account for on the value of the DV. The final SEM will be a Latent Variable

Structural model where hypotheses will be generated to enable the identification of a model that has the best predictive capacity in the context of the study's data.

Path Analysis of the Constructs from SDOM

The inter-correlations amongst independent variables (IVs) and the overall correlation between the IVs and dependent variable (DV) is best illustrated by making use of a Path Analysis diagram. According to Duncan (1966, p. 15), path analysis extends the "verbal interpretation of statistics not of the statistics themselves". It provides a clear indication of the assumptions regarding the ordering of the IVs and the DVs as well as the residual variable that represents measuring error and the influence of unaccounted variables. These factors arguably ensure that any critical analysis of the regression model is sharply focused and relevant not only to the current interpretation but also future inquiry. The path analysis diagram shows the results of the multiple regression model by creating relationships/paths depicted by arrowed lines between the variables together with an indicator of the strength of these relationships. The strength of the relationships are indicated by numerical estimates (beta weights) that emanate from the multiple regression model (Remler & Van Ryzin, 2011, p. 317). The nomenclature used in the path analysis diagram is that the dependent variable is referred to as the endogenous variable and the independent variable is referred to as the exogenous variables.

The initial path analysis model for TASDM, using the study's data as input is illustrated in Figure 6.24.

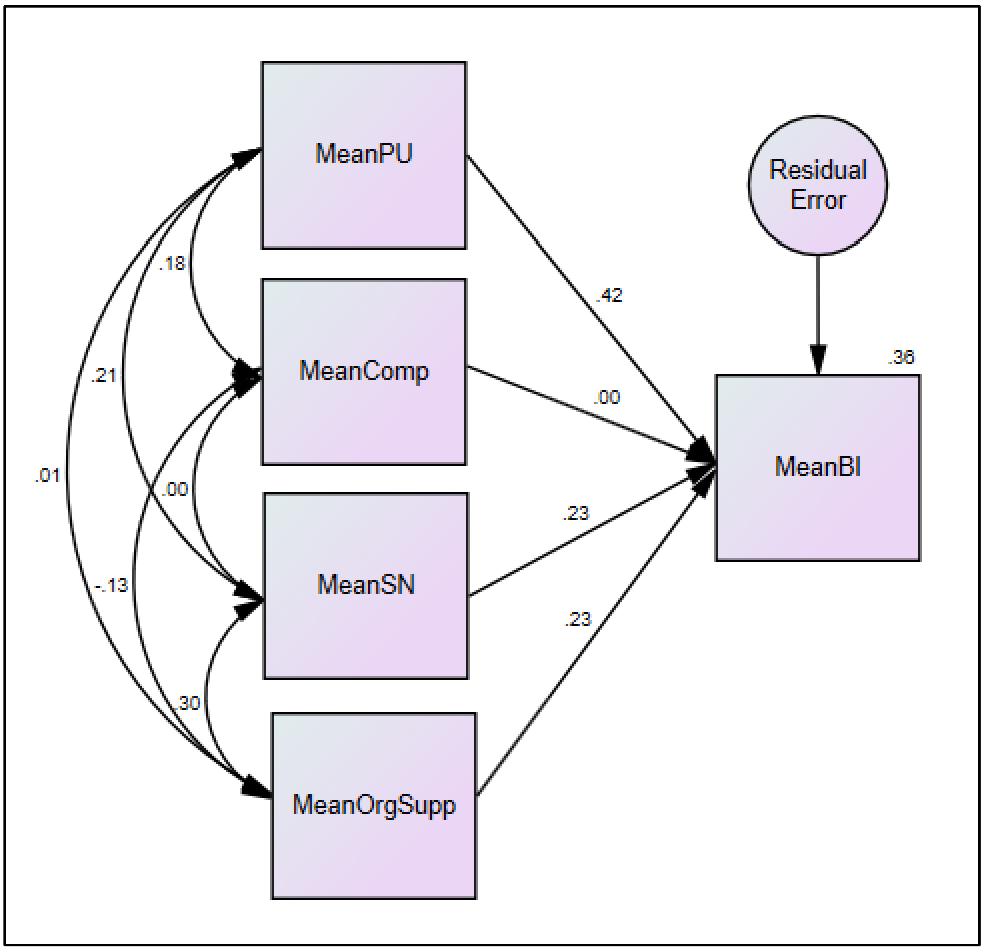


Figure 6.24: Just Identified Path Model for the TASDM

The main outcomes from the Path Analysis diagram illustrated in Figure 6.24 is an affirmation of the results from the multi-regression analysis. TASDM explains 36% of the variance in the respondents' intention (BI) to use SDOM. The main contributors to the predictive capacity of TASDM are PU (42%, p<0.05), SN (23%, p<0.1) and OS (23%, p<0.1). The Residual Error term also referred to as the "disturbance" term represents the composite influence of any other predictive factors that have not been included in the model as well as any measurement errors that may have been committed. The construct of Compatibility (CO) did not make a significantly unique contribution (p>0.05) to the predictive capacity of TASDM in the context of the study's data. The model, based on the measurement data illustrated in Table 6.12, is referred to as a fully saturated or just-identified model where there is a direct path from each variable to each other variable. The fully saturated model may be subjected to a refinement exercise by examining the significance and strength of the relationships between the model's variables. Relationships that are deemed to be insignificant or weak may be removed from the model thereby creating a reduced or over-identified model that is not as complex as the original saturated model. The advantage of developing such a model is that it reduces the fit of the model to the data but also increases its robustness when additional data points are added. However, such an intervention should not compromise the predictive capacity of the just-identified model. Based on the analysis of the values and the significance levels of the variances and co-variances in the just-identified model, the most eligible candidate for elimination is the path from CO to BI and the co-variance relationship between PU and OS, CO and SN and C and OS. A reduced structural model that has been subjected to path elimination is referred to as an over-identified model. The correlation coefficients of the over-identified model for TASDM in the context of the study's data is illustrated in Figure 6.25.

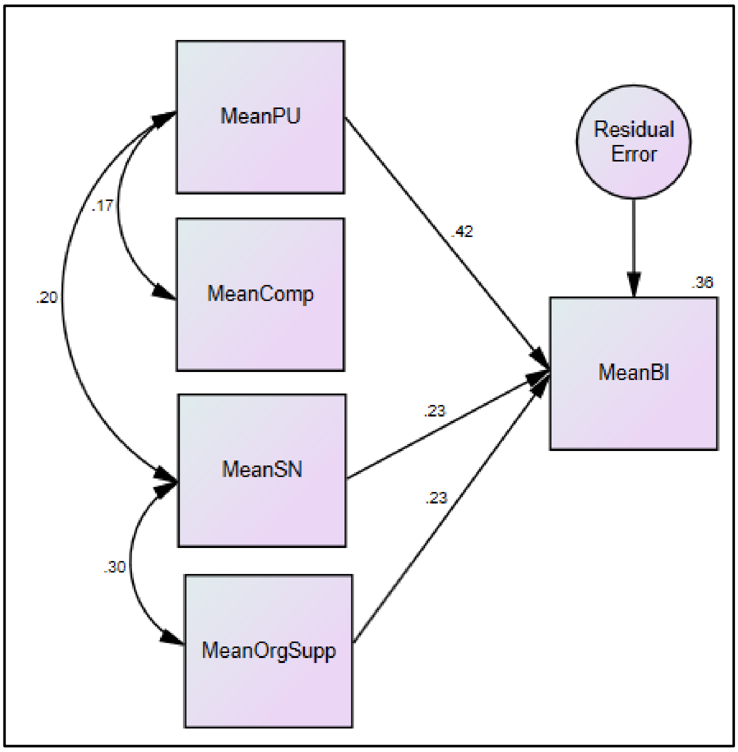


Figure 6.25: Over-Identified Path Analysis Diagram for TASDM

As can be seen in Figure 6.25, the predictive capacity of the over-identified model in the context of the study's data has not been severely compromised when compared to the just-identified (fully saturated) model proposed in Figure 6.24. However, the main predictor variables have been reduced to PU, SN and OS thereby increasing the parsimony of the model. A chi-square (χ^2) test is conducted to determine if the "fit" between the over-identified (reduced) model (illustrated in Figure 6.25) is significantly different from the just-identified (fully saturated) model that was simply a graphical view of the multi-regression model (illustrated in Figure 6.24). According to Hu and Bentler (1999), the chi-square (χ^2) value is traditionally regarded as the most appropriate test statistic that may be used to evaluate the overall model fit as well as the discrepancy between the just-identified model fit and the over-identified model fit. From a hypothesis testing perspective, Hooper *et al.* (2008) advise that the null hypothesis is aligned to the assumption that the over-identified model is a 'bad fit', and a good model fit would produce an insignificant result. In the context of the data from the current study, the over-identified model has been tested to show that there is a significantly (p=0.61) good (not bad) fit between the just-identified model and the over-identified model (χ^2 =0.97, p>0.05).

The path analysis diagram illustrated in Figure 6.25 paves the way for a critical review of the TASDM theoretical constructs in the context of the study's data. The main critical points are listed below:

- There is a lack of contribution from the *Compatibility* construct towards the overall predictive capacity of the model;
- There is high covariance between *Subjective Norm* and *Organisational Support.*

<u>Hypothesised Structural Equation Model (SEM) for SDOM</u>

Whilst the path analysis exercise is pivotal to obtain an illustrative overview of the data and the correlations and co-variances between the main constructs of the theoretical model, SEM enables the researcher to develop hypothesised 'best fit' models for the data and then test the models in terms of their predictive capacity. SEM is a lot more robust than path analysis because it incorporates the original measures of data from the data collection instrument (also called observed or indicator variables) and it attaches an error value to each of these variables. By incorporating the indicator variables into the model, SEM provides the researcher with an opportunity to obtain a deeper understanding of the influence that the indicator variables have on the latent (unobserved) variables (Huck, 2012). The objective of SEM is to provide the researcher with an opportunity to use theoretical knowledge and derive a model through a process of rearrangement of existing variables or to introduce new latent variables and then test the applicability of the model.

From the path analysis exercise conducted on the current study's data, it becomes quite clear that the construct of Compatibility does not have a predictive alignment with the study's data. A possible reason for this phenomenon is the influence of organisational culture where organisations that are perceived to have a hierarchical or group cultural tendency do not endorse innovative behaviour in a dynamic manner, preferring to opt for a cautious approach. Hence, whilst many of the respondents may see the possible usefulness and compatibility of SDOM with agile methodology, they are rather conservative in suggesting that the model has an alignment with their current procedures for software development. Two of the questionnaire items under the Compatibility construct alluded to the relevance of SDOM to the current work-based practice. Both these questions elicited mainly negative responses. Also the disruptive suggestions made by SDOM such as the need for a Build Engineer (BE) and the implementation of automated testing into the sprint cycle may be seen as unwarranted 'disturbances' to a set of development habits that have been established as a norm for many of the respondents. A verbatim comment attesting to the conjecture that the imperative to change development processes is not readily accommodated is made by one of the respondents in the open ended 'comments' section of the questionnaire.

> *We have just moved from the Waterfall approach that has been hard wired into our processes, to a Scrum based approach. That transition was quite slow and painstaking and I'm not sure that we've actually even fully agile as yet. Adding another layer of complexity will take forever to achieve.*

Based on the first phase of data collection and the qualitative analysis, it has become apparent that the influence of organisational culture (OC) is quite strong and has a 'dictatorial' role when it comes to changing behavioural patterns in an organisation. Although the TASDM model is based on social science acceptance theory, it does not incorporate a specific reference to OC. However, the constructs of Subjective Norm (SN) and Organisational Support (OS) have explicit references to the perceived behaviour or attitude of the respondents towards the proposed SDOM in their current organisational setting. These references to perceived behaviour in an organisational setting is further explored by providing a hypothesised model where there is a reconfiguration of the behavioural indicators.

The theory behind this model is that the high measure of covariance between OS and SN observed in the path analysis model from Figure 6.25 is suggestive of a relationship where the perceived organisational support for SDOM is an antecedent to SN. This assertion is corroborated by the regression analysis where it was established that OS is a significant predictor of SN. Based on this evidence, a rearrangement of the relationship between SN and OS is warranted to reflect the antecedent influence of perceived OS on SN. The Compatibility construct has been removed because of its' low alignment with the study's data resulting in a model (Figure 6.26) that displays a tighter coupling between the 3 main constructs from the theory. The main outcome from this exercise is that the predictive capacity of the model has improved from 0.38 (from the path analysis exercise) to 0.58 (explains 58% of the variance in the behavioural intention to use SDOM).

This newly reconfigured model is presented in Figure 6.26.

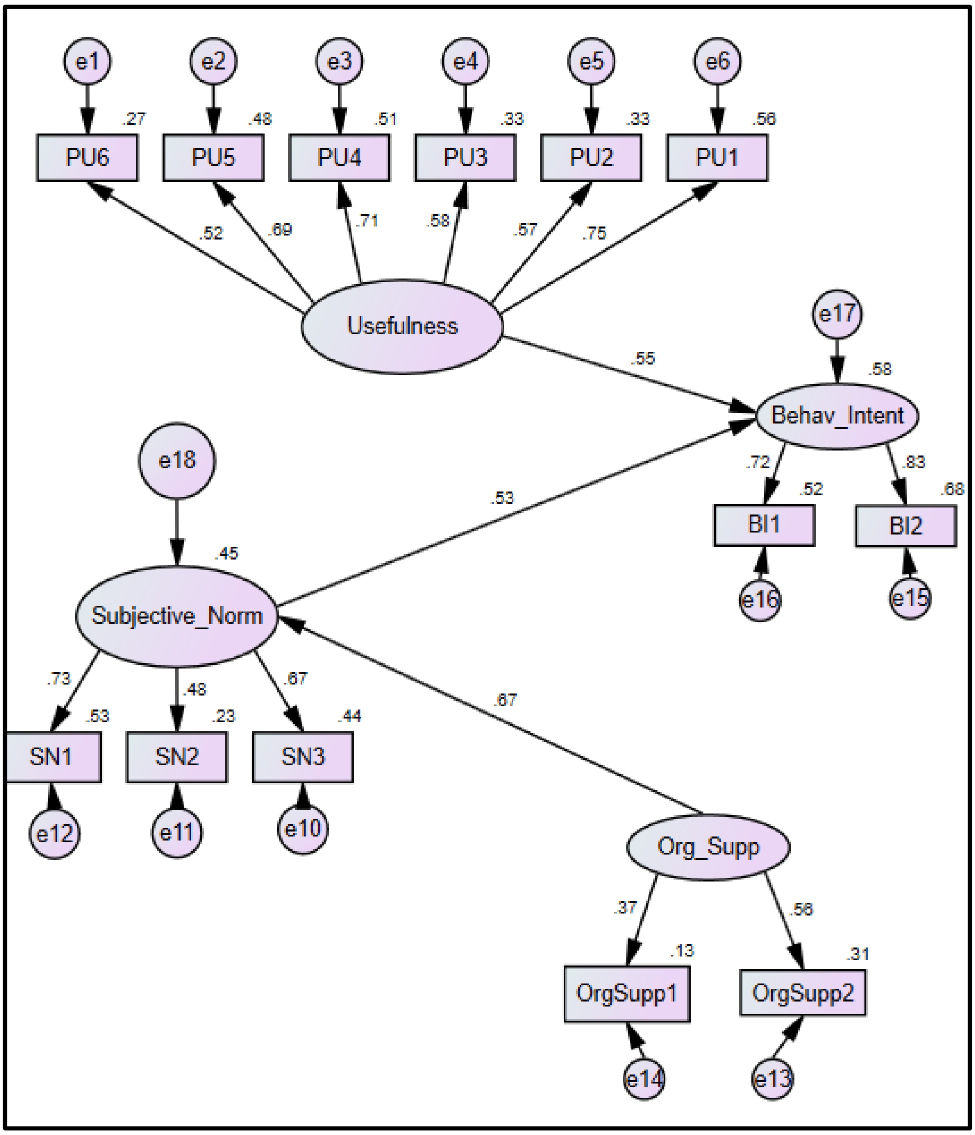


Figure 6.26: Initial Hypothesised SEM for the Study's Data

Although this model seems to have a better and more appropriate fit to the study's data, there is sense of 'lingering doubt' that the model does not capture the full impact of OC. A further hypothesis is proposed that the constructs from SN and OS should be integrated as indicator variables for the latent concept of OC. An addition to this set of indicator variables is the 2nd Likert Scale item from the Compatibility construct where many of the respondents were of the opinion that whilst SDOM did not align to their current work-based practice (items 1 and 3), there was a measure of compatibility (item 2, labelled as Comp2) with the general approach to software development. This outcome of incorporating Comp2 as part of the set of indicators for OC is a second hypothesised SEM that is presented in Figure 6.27.

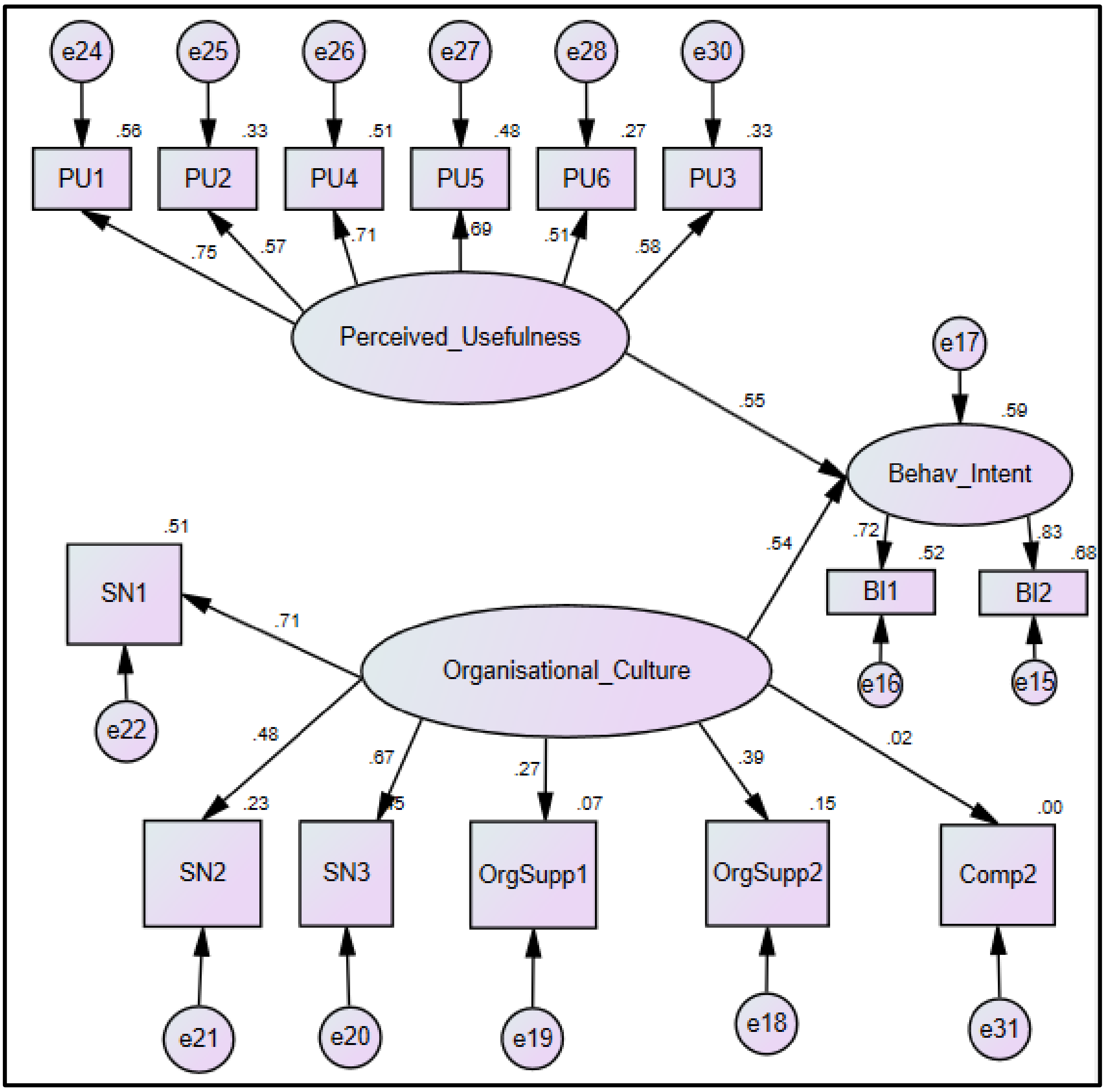


Figure 6.27: Second Hypothesised SEM for the Study's Data

The SEM in Figure 6.27 has a slighter better predictive capacity and now accounts for 59% (an improvement of 1% over the previous hypothesised model) of the variance in the behavioural intention to use SDOM. Another significant outcome is that the influence of OC as a predictor of the intention to use a software development methodology is observed to have increased by 1% as well. A further hypothesis that may be subjected to testing in a subsequent study is that the addition of 'better qualified' indicators of OC to such a model will improve the overall predictive capacity of the model and also magnify the influence of OC whereby it may supersede PU as the main indicator of behavioural intention. However, at this stage a concession has to be made that all the indicators for PU have a significant positive correlation with BI ($p<0.05$). The same claim cannot be made for the indicators of OC where it has been observed that 2 of the OC indicators (Comp2 and OrgSupp2) do not show a significant correlation with BI. Removing both these indicators from the model will enhance the predictive capacity of the model. However, both these indicators contribute as substance indicators to the amorphous concept of OC.

Assessment of the validity of the proposed model is dictated by the parameters for SEM suggested in Byrne (2010). The first significant statistic is the chi square test which yields an insignificant result suggesting that the model has a good overall fit with the data. However, a detailed inspection of the remaining test statistics reveal that the model has a less than 'good fit' with the data. The Goodness of Fit Index (GFI), and the Adjusted Goodness of Fit Index (AGFI) should both be in the range from 0.9 to 1. The GFI and AGFI for the proposed model is reported at 0.86 and 0.83 respectively. The Root Mean Square Error of Approximation (RMSEA) should be less than 0.05. The RMSEA value reported for the current model is 0.063.

The discrepancies and lack of 'model fit' for Figure 6.27 may be attributed to a small sample size and the tenuous adherence to the requirement that the indicator variables should have a normal distribution for the data values. Also, the ad hoc inclusion of the OC dimension may have compromised the fit of the model to the actual data values.

The positive outcome from the proposed model is that it provides a better predictive capacity to understand acceptance and intention to use a proposed software development methodology.

6.5 Discussion of the Quantitative Data Analysis

The objectives of the quantitative data analysis were to:

- Determine the acceptance of SDOM;
- Determine the validity of TASDM and possibly propose a new theoretical model that offers a better predictive capacity to determine acceptance and intention to use a software development methodology.

Discussion of the Acceptance of SDOM

A combination of parametric and non-parametric statistical methods were used to analyse the responses from the study's sample with regards to the acceptance of SDOM. The acceptance of SDOM was operationalised by the constructs from the TASDM model. In order to obtain overview knowledge of the measures of central tendency of each of the 4 constructs from SDOM, the means and medians from the sample responses were computed and subjected to t-tests (parametric) and Wilcoxon one sample ranked test (non-parametric). In the case of the construct of Perceived Usefulness (PU), it was established that the mean response was significant and strongly positive and PU demonstrated the highest significant correlation with the behavioural intention (BI) to use SDOM. This outcome resonates quite well with the results reported in Riemenschneider *et al.* (2002) as well as Ahmad *et al.* (2016). The pivotal role played by PU is also confirmed in general technology acceptance studies such those by Gefen *et al.* (2003) and Schepers and Wetzels (2007), Venkatesh and Davis (2000) and Venkatesh *et al.* (2003),

The result for the correlation between BI and the construct of Subjective Norm (SN) was also significantly positive indicating that the respondents were generally of the opinion that usage of SDOM will be perceived as a positive intervention by people of influence in an organisational setting. The correlation of OS to BI was not significant at the 95% confidence level. However, at the 90% confidence level there is a significant, moderately positive correlation ($p<0.1$). This outcome is not in alignment with the results from the Ahmad *et al.* (2016) study where the significance level is recorded at the 95% confidence mark. A possible explanation for the tenuous influence of OS on BI in the current study is the lack of a better sample size. The result for the construct of Compatibility (CO) cannot be analysed with any degree of confidence and is thus rejected as a significant predictor of BI to use a newly proposed software development methodology. This outcome does not resonate with general acceptance theoretical models. Whilst a general explanation for this phenomenon has been provided in the deliberations regarding the design of the SEM, a further analysis of the lack of significance of this construct is warranted based on its prominence in the annals of theoretical models that explain behavioural intention. An analysis of the questionnaire responses that operationalise the concept of Compatibility reveals that in 2 of the stems (items 1 and 3 of the set of Likert scale items to measure Compatibility), the word 'organisation' is included and the mean response for both these items is classified as negative. However, in Likert scale item 2, there is no mention of the word 'organisation' and the Likert scale prompt simply makes reference to the respondents' opinion on the compatibility of SDOM to general software development practice. The mean response to Likert scale item 2 is marginally positive. These conflicting results would have had a compromising influence on the significance value attached to the Compatibility construct.

In terms of the covariance patterns, there is one discerning observation. There is a strong covariant relationship between OS and SN. A hypothesis that may be ventured is that a perception of good organsational support is an antecedent for a positive disposition towards SN. This conjecture has been tested in the SEM illustrated in Figure 6.26 resulting in an improved predictive capacity of the theoretical model.

Discussion of the Validity of the Theory of Acceptance of Software Development Methodology (TASDM)

The validity of TASDM was tested in the context of the study's data, by making use of CFA and path analysis. The outcome of the CFA exercise confirmed that TASDM had a good overall fit to the study's data, but failed the more discerning goodness of fit tests such as the CFI and TL tests. The path analysis provided an illustrative indication of the predictive capacity of the TASDM which was recorded at 38%. A SEM intervention was employed by rearranging the original latent variables in TASDM and introducing organisational culture (OC) as an additional latent variable. The outcome of this exercise is that the predictive capacity of the model improved to 59%. The significance of adding the OC dimension to the study is that it aligns to the outcome of the qualitative phase of the study where OC was unanimously endorsed as the main determinant of software practitioners' intentions to embrace a new software development methodology.

6.6 Discussion of the Open Ended Responses

The open ended (optional) response section of the questionnaire provided the respondents with an opportunity to make comments and suggestions on SDOM. These responses have been classified as either positive or negative. Twenty-two of the respondents provided written feedback on SDOM. Eighteen of these responses were interpretively classified as positive responses with the remaining four being classified as negative.

Positively Worded Responses

The positively worded responses were not too informative. The common theme with these responses were that SDOM does herald an improvement to Scrum based agile development. A verbatim response made by an experienced developer in the banking sector is provided as a source of reference.

> *The good thing about SDOM is that it breaks the silo mentality. It ensures that there is interactivity between the developers and the operations staff without adding too much of complexity to the sprint cycle.*

These comments epitomise the objective of SDOM. Another comment made by a project manager/team leader from the bespoke software development domain that has a similar thematic alignment with the previous comment reads as follows:

> *...going forward however we do plan on increasing our staff number and making roles more defined and I do think that having ops members knowing what is involved and happening in a dev release is vital. I have as a project manager (in this job and previous ones) always made a point of informing the operational staff of what was involved in a release so that they weren't going into an install blind. They have to know what functionality is being installed so that they know if things are working or not - if things fail they can troubleshoot and determine if it's something they can sort out before escalating it to the dev guys.*

This comment resonates with the imperative to mitigate the problems that may arise because of a lack of communication between the development team and the operations team and ensure that there is a seamless rollout of software along the development, delivery and deployment pipeline. Included with the positively worded comments were however a few words of caution. The following comments were extracted from the open ended response by a software developer who specialises in security and testing at a financial organisation.

> *If this (model) has to be customized to our environment, one thing that has to be added is clarity on how the "production-like" environment relates to SIT which is an area used for testing that is as close to production as possible*

In the extract, SIT is used as an acronym for security, integration and testing. A concession that has to be made is that SDOM does not have any specific reference to the security imperative which has become part of the core focus for modern software systems. During the course of the current study, the researcher has been co-opted as a member of the IEEESA Working Group on agile software development where fellow group members delivered a paper that addresses the issue of security in the agile development environment. The paper by Yasar and Kontostathis (2016) provides a lightweight DevOps model (Figure 6.28) that has the objective of providing a specific security and testing focus during the agile development lifecycle.

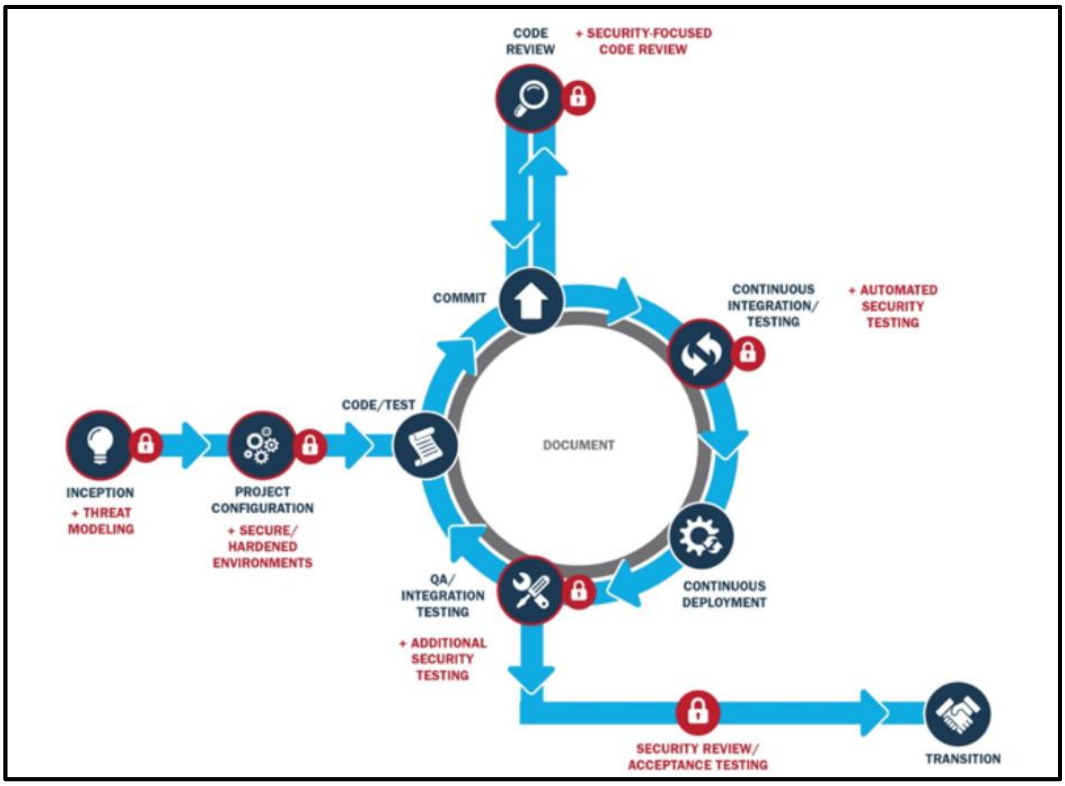

CODE REVIEW
+ SECURITY-FOCUSED CODE REVIEW
COMMIT
CONTINUOUS INTEGRATION/ TESTING
+ AUTOMATED SECURITY TESTING
CODE/TEST
DOCUMENT
INCEPTION
+ THREAT MODELING
PROJECT CONFIGURATION
+ SECURE/ HARDENED ENVIRONMENTS
QA/ INTEGRATION TESTING
+ ADDITIONAL SECURITY TESTING
CONTINUOUS DEPLOYMENT
SECURITY REVIEW/ ACCEPTANCE TESTING
TRANSITION

Figure 6.28: A Security and Testing Model for DevOps proposed in Yasar and Kontostathis (2016)

The basic tenet of this model is that the error situation of 'breaking the build' that has become synonymous with the strategy of continuous integration (CI) should not have an exclusive focus on functionality, but also on security. SDOM does make provision for testing, both in a manual and automated form. A lightweight addition to SDOM could be a specific reference to security based testing. However, the Yasar and Kontostathis (2016) model makes a call for a specialised security testing team to be incorporated into the DevOps development lifecycle so that there are specific security testing phases that are attached to the lifecycle model. The methodology suggested by Yasar and Kontostathis resonates with a call previously made in Arkin *et al.* (2005) that software development lifecycle models should incorporate a penetration testing phase eloquently described by Geer and Harthorne (2002, p. 1) as "…the art of finding an open door" or a test to determine the vulnerability of a system to unauthorised access from the point of entry to the availability of functionality that is designated for specific users of the system. The model proposed by Yasar and Kontostathis suggests that security testing teams should be involved in the daily build cycles so that a 'break of the build' could be attributed to security breaches that could be identified as the system evolves rather than at the end of the development lifecycle. As illustrated in Figure 6.28, the security testing team has to be involved at the system inception phase, during the daily code commits and once the system is configured so that it is in a 'production-like' state. At the inception stage, the security team is responsible for developing a document that outlines the penetration testing strategy and this document is updated, based on the outcomes of the various penetration testing interventions conducted during the daily and production build cycles.

A closer examination of the Yasar and Kontostathis model reveals that the model is not as complex as initially perceived and an integration of this model and SDOM can be easily achieved. This will provide an ideal security oriented enhancement to SDOM.

Negatively Worded Responses

The negatively worded responses resonated with the opinion that SDOM did not align with some of the mainstream approaches that make an organisation-wide impact and it is not an international standard that will contribute to any planned level of agile maturity. The researcher provided email responses (refer to Appendix F), to the respondents who provided the negative responses. The content of the email communication was essentially to endorse the validity of the negative comments and also to explain that these comments were certainly applicable to organisations where the culture was amenable to an organisation-wide agile and DevOps intervention. Two of the negatively worded verbatim responses are included for reference.

> *The differentiation and advantages of SDOM over these mainstream frameworks needs to be clearly articulated and quantitatively verified with actual metrics on real software projects that many organisations are aspiring towards…not sure if I would adopt SDOM if out of the box frameworks like* ***SAFe*** *are available which provide training and*

> *certification for their frameworks.* **(Software engineer for a national logistics organisation)**
>
> *This is a "light-weight" SCRUM Delivery Model and I unfortunately don't see benefits of SDOM than what is already provided by the originally authored model. I would suggest a model that emphasizes more on quality and reuse of existing / strong frameworks that would assist agile delivery.*

(Software engineering consultant for a national/international software provider)

The reference to the Scaled Agile Framework (SAFe) is certainly valid. However, the researcher is of the opinion that SDOM may be seen as a subset of the SAFe. The respondent who made this comment did agree during the subsequent email conversation that the SAFe required a major organisational cultural shift that could entail an intensive investment of resources and organisational time that is not easily achieved and sustained. The 'second negative respondent' was assured that SDOM does have a strong Scrum alignment but it has been tweaked to incorporate a 'DevOps flavour' thus providing a technical differentiation between SDOM and a pure Scrum based model. The reference to reuse and the reliance on existing development frameworks was clarified in a follow-up email to allude to a software engineering strategy currently being touted as model based software engineering (MBSE). The MBSE is an initiative that encourages the use of tried and tested software code bases and design models that have been previously used as solutions to business problems. The MBSE is seen as an attempt to regulate/standardise the software industry with tried and tested models of development. It does however rely on a comprehensive model set to work from. The lightweight nature of SDOM enables easy integration with software strategies such as MBSE and this could be a consideration for subsequent versions of SDOM. It also requires a heightened security add-on because many of the pre-defined models will be sourced from 3rd party vendors. This ties up with the security and testing model proposed by Yasar and Kontostathis (2016). The strength of SDOM is manifested in its lightweight capacity enabling it to integrate easily with software engineering, operations engineering and security improvement initiatives.

The main outcome of the open ended deliberations is that the discourse on methodology models such as SDOM will invariably provoke passionate responses that are diverse but add to a constellation of ideas that contribute to the evolutionary trajectory of software development methodology. This trajectory provides an enabling environment for the attainment of maturity of software development processes that could culminate in the universal acceptance of SAFe-like frameworks where software development assumes a ubiquitous presence in organisational processes.

7.0 SUMMARY AND CONCLUSION

7.1 Introduction

This chapter comprises of the study's summary and conclusion. The summary is structured by making a reference to the study's research questions with an accompanying discussion of how these questions were answered. The chapter outline is as follows:

- A review of the research objectives and questions underpinning the study;
- A discussion of the study's findings;
- Theoretical contributions of the study;
- A review of the study's limitations;
- The implications arising from the study and areas for future research;
- An autobiographical reflection;
- The study's conclusion.

7.2 A Review of the Study's Research Questions

The overarching objective of the study is to make a contribution to the field of software engineering (SE), specifically in the domain of software process methodology (SPM) and software process improvement (SPI) strategies. The attainment of this objective is guided by a main research question that has underpinned the study:

How can experiential knowledge of Agile Software Development practice in South Africa be used to develop a Socio-technical Framework to Guide the Implementation of Agile Software Development Methodology?

The essence of this question is attributed largely to the outcome of the study's literature review as well as an interview conducted by the researcher with IBM Research Fellow, Grady Booch. The outcome from both these sources converge to an opinion that the study of SPM has to be conducted from a social and a technical orientation. The social dimension manifests in the experiential knowledge of SPM by expert software practitioners. One of the major influences that makes a pivotal contribution to the choice and implementation of a SPM is the culture that prevails in an organisation. The technical dimension is a reference to the engineering-like methods that are intrinsic to the software development process. The evolution of software process methodologies is following a trajectory that is gravitating towards the iterative and incremental approach to software development. The allure of the iterative and incremental approach is that is embodies flexibility and it has an ability to deliver working software quickly. The attributes of speed and agility have been aggregated into a set of methodologies referred to as Agile Software Development Methodology (ASDM). The main objective of the study is to provide a social and technical intervention that will enhance the implementation of ASDM. The empirical phase of the study entailed the acquisition of experiential knowledge from software practitioners in South Africa to enable the study to meet this objective.

In order to leverage the knowledge of experienced software practitioners from a meaningful, 'depth-oriented' perspective the dominant methodological approach implemented in the study is qualitative, with a specific adherence to the principles of phenomenology. The empirical phase of the qualitative component of the study comprised of a set of 16 semi-structured interviews conducted with South African based software practitioners who have acquired general software development experience as well as ASDM experience in an organisational context. A content analysis of the interview data was performed to enable a synthesis phase that comprised of 2 models with each having either a social or a technical orientation. The socially oriented model developed in the study provides a framework that integrates the implementation of ASDM with attributes of organisational culture (OC). The technically oriented model developed in the study is an extension of one of the prominent agile methodologies so that it is better aligned to the requirements identified by the cohort of software practitioners who were interviewed. The final empirical phase of the study entailed a quantitative verification exercise to establish practitioners' acceptance of the *technically* oriented model. Technology acceptance theory was used as a platform to identify an appropriate theoretical model to underpin the quantitative verification phase of the study. A total of 40 experienced software practitioners were surveyed to establish their acceptance of the proposed technical model to guide the implementation of ASDM.

The main research question is broken up into the following sub-questions:

- What are South African software practitioners' perspectives on Agile Software Development Methodology (ASDM) from a technical perspective?
- How does organisational culture influence the implementation of ASDM?
- How can South African software practitioners' knowledge of ASDM be used to develop a framework to guide the implementation of agile methodology?
- What is the acceptance by South African software practitioners of a framework that informs the technical implementation of ASDM?

7.3 A Discussion of the Study's Findings

The study's findings will be presented fractionally by making reference to the sub-questions that guided the study. The discussion of the study's findings will culminate in a holistic review of the main outcomes from the study.

What are South African software practitioners' perspectives on ASDM from a technical perspective?

All 16 interviewees endorsed ASDM as a relatively successful methodology for software development. This response was made in comparison to the Waterfall methodology which many South African organisations were either phasing out completely or were in the process of transitioning to ASDM. The underlying imperative for the migration from a prescriptive, plan driven approach to a dynamic, agile approach was the perception that system development occurred a lot faster, thereby enabling the attainment of business value. Also, the development process was flexible enough to accommodate changing user requirements resulting in greater satisfaction by the system's stakeholders. The practitioners were also of the opinion that the agile approach enhanced morale within the team of developers and promoted a sense of ownership and accountability. This outcome resonates well with similar reports regarding the benefits of adopting an agile approach to software development (e.g. (Abrahamsson *et al.*, 2017; Dybâ & Dingsoyr, 2008; Moniruzzaman & Hossain, 2013; Nguyen, 2016).

A defining trend was that all interviewees explained that the adoption of agile methodology entailed a migration from a Waterfall based approach to a *Scrum* based approach for software development. The adoption of Scrum as the current de facto methodology for software development in South Africa is strongly aligned to global trends with regards to a preference for a specific agile methodology (as reported in (Dingsøyr & Lassenius, 2016; VersionOne, 2016) There was however, a substantive effort made by all the members of the cohort of practitioners who constituted the interview panel, to explain the need to implement a customised version of Scrum. The customisation consisted of a hybrid of XP and Scrum oriented methods. The Extreme Programing (XP) methods that were invoked are test driven development, code refactoring, continuous integration and pair programming (to a lesser extent). The Scrum methods that were invoked are the maintenance of a product backlog, the Scrum Sprint phase, the daily standup meetings and the integration and system testing and release of the system into a production environment. These practices resonate well with the findings by Abrahamsson *et al.* (2017) that the Scrum/XP hybrid arrangement typically consist of XP methods that provide guidance on the engineering aspect of software development and the Scrum methods provide guidance on the project management aspect of software development.

A deviation from the agile approach does however manifest in the preference by South African organisations to engage in a big design upfront (BDUF) strategy. This is contrary to the dictates of ASDM where the upfront design is specified as a high level design effort with the detailed design evolving with the coding of the system. However, practitioners expressed reservations about this practice and intimated that the smaller, less critical projects did adopt a high level, minimalist upfront design strategy aligned to the agile approach. However, for the larger organisation-wide applications, a BDUF strategy was employed because this strategy enabled better project management of the system. This Waterfall based customisation of ASDM resulted in the use of the term Wagile development that entailed a BDUF followed by a Scrum approach for the coding of the system. This strategy was also aligned to the culture in the organisation where there was a preference to forego competitive advantage in order to preserve a predictable software development process.

In a majority of the interviews (69%), there was a reference to the issue of business value. The inability of the Waterfall methodology to deliver on business value was touted as one of the reasons that lead to the demise in the popularity of the Waterfall methodology. Although the migration to ASDM and Scrum has alleviated the situation somewhat, this dilemma was still largely unresolved. The Scrum model has been criticised for not providing direction to enhance the scalability of the solutions developed to an organisation-wide platform.

The overriding conclusion from the practitioner perspective is that there is a preference for ASDM and the ideal arrangement is a Scrum based methodology infused with Waterfall and XP methods. However, this Scrum oriented approach was ideal for developing ‘solutions in the small’ but did not scale up to level where the solution could be directly implemented on the organisational infrastructure.

This setback resulted in the late delivery of expected business value that could be derived from the system.

The issue of the scalability of agile methodologies such as Scrum has only recently been receiving attention in the academic literature. Dingsøyr and Lassenius (2016) suggest that the adaptation and customisation of agile methodologies have been widely recognised by the academic community and the practice of customising ASDMs like Scrum have also become entrenched into the practitioner domain. However, there has been a severe lack of focus by the academic community on the scalability of ASDM (an assertion that has been endorsed by Dikert *et al.* (2016) and Fuggetta and Di Nitto (2014)). According to Dingsøyr and Lassenius, the lack of direction on the scalability of ASDM is one of the major shrtcomings of the methodology. This assertion is corroborated by the empirical evidence gathered on the basis of the analysis of South African practitioners' perspectives of ASDM in the current study.

How does organisational culture influence the implementation of ASDM?

The inclusion of OC as a structural component of the current study has been necessitated by the pivotal role that it plays in the adoption of a software development methodology. The study of the influence of OC is not easy to achieve because of the difficulty of acquiring an oversight role over an amorphous concept such as OC. In order to overcome this problem, the study implemented the Competing Values Framework (CVF) to operationalise OC. The CVF classifies OC along dimensions that range from a high level of internal control that is associated with a hierarchical style of management control to lower levels of internal control and a more democratic management approach.

From a holistic perspective, the results of the data analysis indicate that there is unanimous agreement that OC influences the adoption of ASDM, thereby confirming the reports from the academic literature (Iivari & Iivari, 2011) on this phenomenon. However, the classification of OC along the dimensions of the CVF is not a straight forward process. The cultural mix that exists in South African organisations is varied with a predominance of the hierarchical management style (referred to as Hierarchical Culture from the CVF) that prevails at the upper echelons of management. From a technology perspective, this hierarchical culture was also prevalent at the lower/supervisory management levels during the tenure of the Waterfall methodology because each phase of the development process required management approval before the next phase of development could commence. However, the migration to an agile approach has resulted in a more organic management style where the focus is on co-operation and consultation and the ability to respond to change without becoming too embroiled in bureaucratic processing. Agile teams work in a self-managing environment and this sense of autonomy enables agile teams to achieve high levels of productivity. From a CVF perspective, a Developmental Culture orientation (which encourages leadership and collaboration, innovation and risk taking in the quest to achieve competitive advantage) has a strong resonance with ASDM. The CVF also makes reference to a Group Culture orientation (driven by experienced staff members who provide a flexible environment but prefer to have internal control) which also has a resonance with ASDM. Cockburn and Highsmith (2001) make reference to the requirement that ASDM can only be successful in an "agile organisation" that consists of "agile managers" (p. 132). The current OC classification provides clarity on these assertions by identifying 4 'quadrants of applicability' for ASDM.

Identification of the 4 'quadrants of applicability' for ASDM within an OC framework is based on empirical references (9 of the 16 interviewees (56.25%)) to a phased-in or a bimodal approach to software development where software development teams are cajoled into an agile environment from a Waterfall foundation. This approach is guided by the experienced staff members who prefer to have a measure of control as organisations migrate from a Waterfall to an agile environment. This migration is coupled with a change in management style resulting in a shift from a 'command and control' style of management, which are symptoms of a Hierarchical Culture, to a collaborative Group Culture orientation. Based on the empirical evidence in the study, organisations in South Africa have a predominant Hierarchical Culture (previously confirmed in a study by Iivari and Huisman (2007)). However, there is an imperative for these organisations to migrate towards a Developmental Culture orientation where software development teams are entrusted with complete autonomy so that the invocation of ASDM would enable the delivery of greater business value to the organisation. The empirical evidence suggests that this migration is currently being achieved in a measured approach where organisations are 'breaking out' of the 'hierarchical mould' by venturing into a mix of Group Culture and Rational Culture orientations. This arrangement is not ideal because the most fertile region for ASDM to deliver on its expectation of business value is in the Developmental Culture dimension (Iivari & Iivari, 2011). However, the transition to a Developmental Culture requires an organisation-wide cultural shift that comprises of a closer collaborative environment between the business and technology divisions where technology is the driver of business value. The transition to Developmental Culture also entails the bestowing of complete trust onto the development teams to arguably ensure that business value is delivered and failure is mitigated by comprehensive risk analysis and testing. These shifts in organisational behavioural norms can only be achieved by providing comprehensive training programs that empower software developmental teams with decision making acumen from a technological and *business* perspective. The corollary of this arrangement is to empower business analysts (BA's) with knowledge of the software development process so that the environment between software development teams and business managers is more collaborative rather than command and control.

Based on the input obtained from interviewees from 2 of the major banking institutions in South Africa, the attainment of Developmental Culture status is something that these institutions are aspiring towards. The main reason for this imperative is the realisation that the lack of scalability of ASDM is resulting in a loss of the expected business value. In order to mitigate this situation, these organisations are resorting to scaled versions of ASDM such as the Scaled Agile Framework (SAFe). However, this transition will require a business process re-engineering initiative coupled with organisation-wide training on SAFe that may not be easy to achieve. The remaining 2 banking institutions are content to abide by a Group/Rational cultural mix with a focus on trying to mitigate the agile scalability dilemma with lightweight adaptations of Scrum.

In conclusion, OC influences the adoption of ASDM. The least 'fertile' region for ASDM is the Hierarchical Culture dimension of the CVF. The ideal placement of ASDM is in the region of Developmental Culture (from the CVF). The migration from Waterfall methodology to ASDM in South Africa has been coupled with a change from a Hierarchical Cultural environment to one that resonates with the Group/Rational culture mix. This arrangement does not enable the attainment of maximum value from ASDM. However, it is a workable compromise that facilitates the transition to ASDM without having to engage in a complete organisation-wide cultural shift as is required by SAFe. Based on input from South African software practitioners, the typical OC attributes of organisations that are migrating towards ASDM is that innovation and collaboration is highly valued, but there is a strong tendency to uphold current organisational traditional practices and norms. These attributes place these organisations at an overlap region between the Group Culture and Developmental Culture classification from the CVF. Hence, any intervention to improve the implementation of ASDM in these organisations should ideally be located in the intersection region of the Group and Developmental Culture quadrants of the CVF.

How can South African software practitioner's knowledge of ASDM be used to develop a framework to guide the implementation of agile methodology?

The current research question is a reference to the synthesis phase of the study. The synthesis phase comprises of a social and a technical model that have been derived as output elements of the qualitative analysis conducted on the interview data. From an overview perspective, one of the more conspicuous trends with regards to the adoption of software development methodology in South African organisations is that the 2 most popular methodologies are the Waterfall and Scrum methodologies. The trend observed is that organisations are making an effort to migrate from a Waterfall methodology to the Scrum methodology. This migration is conducted by adopting a phasing-in approach that embraces a customised version of the Scrum methodology. The 2 main areas of customisation is the amount of BDUF that is conducted and the level of project management that is maintained during the development process. Based on the preceding narrative, an agile version of the CVF has been developed that classifies the type of development methodology that best matches each of the 4 OC classifications in the CVF. The model classifies OC along a continuum from a high level of control and low levels of flexibility to low levels of management control and a high level of flexibility. The Waterfall methodology is matched to the Hierarchical classification of OC. However, at the Group Culture level where management control is not prescriptive and imposing, but driven more from a guidance perspective, the Wagile (Waterfall/Scrum) approach is recommended. This approach allows the more experienced development team members to dictate the amount of BDUF that may be conducted and customise the level of agility according to the organisational norm or the requirements of the project. At the Rational Culture level where the focus is on process optimisation and efficient control of resources, a risk oriented software development methodology is suggested. The ideal candidates for the Rational Culture 'quadrant' in the CVF is the Spiral methodology (Boehm, 2006) or the updated, agile oriented version of the Spiral model named the Incremental Commitment Spiral Model (Boehm, 2011). At the Developmental Culture level of the CVF, a comprehensive adoption of ASDM is suggested. At this level, the all-encompassing SAFe framework that mandates an organisation-wide adoption of the principles of agility, is suggested as an ideal approach for software development.

The analysis of empirical data revealed that a substantive shortcoming of ASDM is the lack of scalability of the methodology thereby compromising the attainment of business value. The issue of scalability has been traced to a silo-based approach to software development that is prevalent in most organisations. The symptom of the silo-based approach is the lack of collaboration between the business, the software development and operations divisions. The business analyst (BA) serves the role of providing a conduit that conveys business requirements to the software development division. However, once the software has been developed, there is a lack of collaboration between the business stakeholders, the software development team and the operations team that is entrusted with the main non-functional requirement of ensuring that the system integrates with the organisational infrastructure. An outcome of the data analysis is that there is a need for a Build Engineer (BE), who provides the linkage between the development team and the operations team as well as the business division. The BE will have the responsibility of ensuring that there is a seamless transition from business requirements to systems development and deployment onto a 'live' organisational platform. Basically, the BE serves the role of ensuring that the system is in state of readiness for activation to a live production environment thereby ensuring that the attainment of business value is not compromised by a time delay between development and operations. Currently the Scrum methodology does not have any operations functions interwoven into the Scrum 'ceremony' of development. As a solution to this problem a model has been developed in the current study, named the Scrum Development Operations Model (SDOM). SDOM has a Scrum infrastructure infused with operations activities controlled by the BE. This model provides a forum for the BE to arguably ensure that the development environment is configured as close as possible to the operations environment where the system will be deployed. The main modification made by SDOM is the traditional definition of 'done' (from an agile nomenclature perspective) is adjusted so that it alludes to a state where the system has been developed and tested from a functional perspective and also tested from an infrastructure (non-functional) perspective so that 'done' now refers to a system that is in a deployable state. SDOM consists of operations activities juxtaposed onto the traditional Scrum model as a response to the shortcomings of the latter model identified during the analysis of the study's empirical data.

What is the acceptance by South African software practitioners of a framework that informs the technical implementation of ASDM?

In order to obtain a perception of SDOM, a quantitative survey of 40 purposively selected software practitioners was conducted. The selection criteria used was that the practitioners must have at least 5 years of experience in general software development and at least 2 years of experience in the use of ASDM. The group of software practitioners who formed the main cohort of 16 interviewees from the first empirical phase of the study was also included in the sample. The questionnaire used to establish acceptance of SDOM was informed by an adapted version of the Riemenschneider *et al.* (2002) Theory of Acceptance of Software Development Methodology (TASDM). The main constructs of the theoretical model are Perceived Usefulness (PU), Compatibility (CO), Subjective Norm (SN), Organisational Support (OS) and Behavioural Intention (BI).

The results from the quantitative analysis indicate that 80% of the respondents had a positive disposition towards the PU of SDOM. The construct of PU was also found to be the strongest predictor of the BI to use SDOM and accounted for 42% of the variance in BI. In the case of SN, 60% of the respondents had a positive disposition towards SDOM indicating that the majority of the respondents felt that the use of SDOM will be endorsed by 'people of influence' in an organisational setting. The construct of SN was also found to be a significant predictor of BI to use SDOM and accounted for 23% of the variance in the BI construct. In the case of OS, 62.5% of the respondents had a negative disposition towards SDOM and were of the opinion that their organisations would *not* provide management and resource support for the implementation of SDOM. The construct of OS was *not* found to be a significant predictor of BI to use SDOM. In the case of CO, 67.5% of the respondents had a *negative* disposition towards SDOM indicating that the processes contained in SDOM were not compatible with the software process currently implemented in these organisations. The construct of CO was *not* a significant predictor of BI to use SDOM. From a holistic perspective, 80% of the respondents had a *positive* disposition towards a BI to use SDOM.

The empirical evidence from the current study attests to the influence of PU as the strongest determining factor of BI to use SDOM. This outcome is aligned to the influence of PU on BI in many other empirical studies regarding adoption behaviour (e.g. Adams *et al.*, 1992; Agarwal & Prasad, 1998; Davis, 1989; Riemenschneider *et al.*, 2002; Saadé & Bahli, 2005; Venkatesh & Davis, 2000). This trend is also confirmed in meta-analysis studies of adoption behaviour such as King and He (2006) and Schepers and Wetzels (2007). The empirical evidence also attests to the relatively lesser influence that SN has on the BI to use SDOM. This outcome is aligned to the reported influence of SN in studies of adoption behaviour such as Venkatesh and Davis (2000), Vijayasarathy and Turk (2012) and Schepers and Wetzels (2007).

According to Riemenschneider *et al.* (2002), PU and SN are the 2 most decisive factors that influence an individual's decision to adopt a software development methodology. The software practitioners who have been sampled in the current study indicate a positive disposition towards the PU and SN of SDOM. The argument presented qualifies the deduction that software practitioners perceive SDOM to be a useful model that will be positively viewed by their peers in the software development domain.

The detractor to the deduction made in the preceding paragraph is the negative disposition towards Compatibility and Organisational Support towards SDOM. Both these constructs, Compatibility (Hardgrave *et al.*, 2003; Riemenschneider *et al.*, 2002) and Organisational Support (Ahmad *et al.*, 2016) were reported as significant contributors towards a decision to adopt a software development methodology. In terms of general technology and innovation adoption theory Compatibility has also been reported to be a significant contributor towards the BI to adopt a technological innovation (Kai-ming Au & Enderwick, 2000; Lee *et al.*, 2011). A closer inspection of the Compatibility construct does however reveal that the organisational context may have a confounding influence on the values recorded for this construct (Agarwal & Prasad, 1998; Fichman, 2000; Mustonen-Ollila & Lyytinen, 2003). The organisational context alludes to either the technological infrastructure (Agarwal & Prasad, 1998) or the organisational culture (Fichman, 2000). According to Fichman (2000) the adoption of an innovation in an organisational context entails a lot more complexity than accorded by classical diffusion theory where the Compatibility construct is measured by determining whether the innovation is aligned to current work practice. Karahanna *et al.* (2006) suggest that compatibility is a multivariate construct and should be decomposed to measure compatibility with existing work practice and compatibility with an individual's personal value system and preferred method of work.

In the context of the current study, the items used to measure compatibility was aligned to classical diffusion theory, thereby not accounting for the compounding influence of organisational culture. Many organisations exhibited symptoms of Hierarchical or Group culture, suggesting that the introduction of innovation follows a bureaucratic process of adoption that is controlled by the organisation's management. Hence, the compatibility of SDOM with the current work practice of the cohort of respondents may have been compromised by the perception that the proposed model may not receive management support. This phenomenon may also explain the lack of convergence of the Organisational Support construct to the theoretical model used to underpin the questionnaire design. The anomalous data associated with the constructs of Compatibility and Organisational Support also highlight the intricate link between the technical elements of the proposed model and the social context in which it may be used. One of the discerning aspects of SDOM is that it proposes that the BE plays an integral role in the Scrum planning and development cycle. This 'disruptive' intervention may not be fully aligned with the systems development team structure currently used in the organisations. In organisations where the culture is deemed to be a Hierarchical or Group culture, such a reconfiguration of the development team structure and rearrangement of Scrum practices such as the coding, integration and testing phases may require extensive deliberation by management before such a change could be sanctioned. However, in an organisation that exhibits a Developmental culture, the 'barriers of resistance' and the intensity of bureaucratic deliberations are not that great when it comes to embracing innovative suggestions such as SDOM.

The study's *main* research question is presented again for reference.

How can experiential knowledge of agile software development practice in South Africa be used to develop a socio-technical framework to guide the implementation of agile software development methodology?

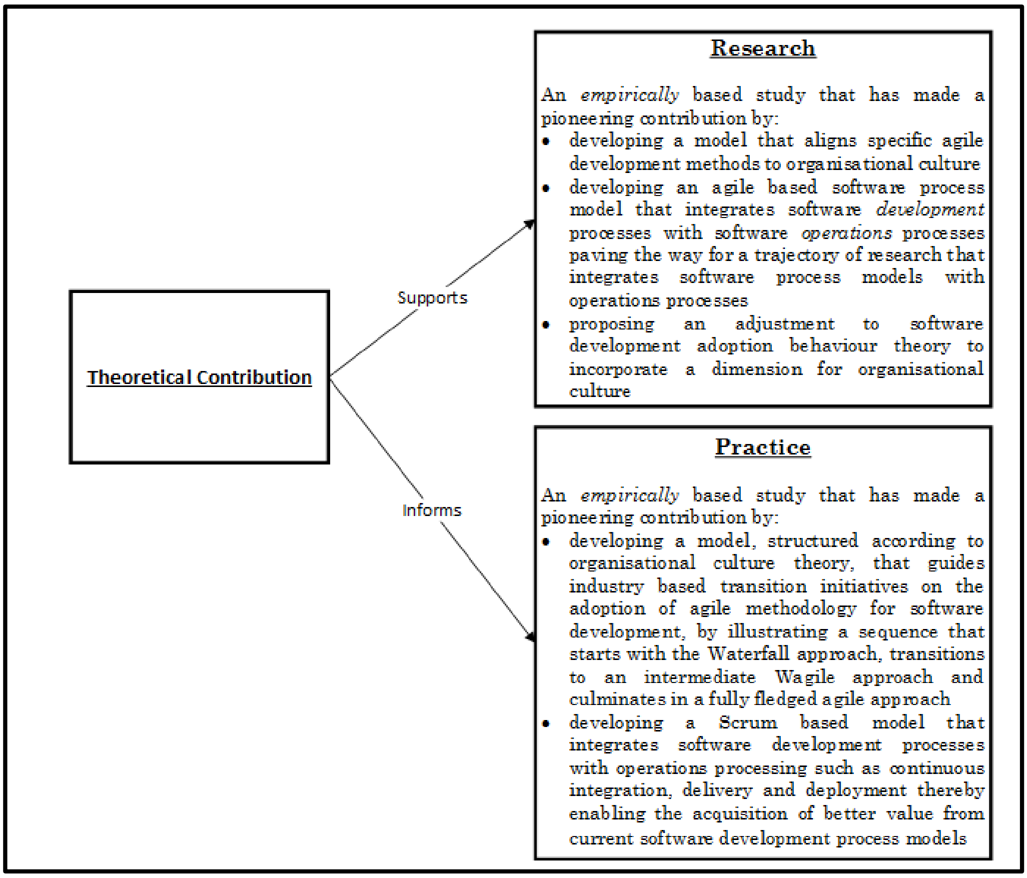
The study has answered the main research question by leveraging experiential knowledge of software practitioners to propose 2 models (the agile based OC framework model Figure 5.13 and the Scrum Development Operations Model in Figure 5.18) that provide guidance on the implementation of ASDM. The social dimension is represented by a model that aligns OC to a compatible methodology for software development. The model provides a framework that is structured to inform the migration to agile methodology along the dimensions of change in OC. The culture of an organisation is entrenched into the work practice of the employees over a period of time and changes to this culture cannot be implemented in a short time period. Aligned to the imperative of many organisations in South Africa to migrate towards an 'agile friendly' culture, there is a need to adopt software development methodological approaches that provide the expected benefits of agile adoption in an incremental, lightweight manner that does not necessitate 'sweeping' changes to the culture of an organisation. The SDOM model has been proposed so that it aligns with the current technical implementation of agile methodology and requires only a marginal change in the OC.

Although both the models proposed in the study are conceptually different and convey a dichotomous relationship, there is an underlying intricacy that links both models. The OC model has an inextricable link to the adoption of technically oriented software development process models and SDOM has been crafted to embody a simple and lightweight deviation from traditional agile methodology so that it easily aligns with the intersection of the Group and Developmental Culture orientation that is typically found in South African organisations.

7.4 Theoretical Contributions of the Study

As Gorla and Lin (2010) as well as Sjøberg *et al.* (2008) point out, there is little consensus in academia as to what constitutes a theoretical, scholarly contribution to a field of study. Sjøberg *et al.* (2008) do however, provide some guidance in terms of scholarly theoretical contributions in the domain of software engineering (SE) by suggesting that such theory should be aligned to the philosophy of pragmatism because SE is an applied science and should ultimately benefit industrial practice. In order to achieve this objective, the theoretical contribution should be delineated according to the dimensions of research and industrial practice. The research component should contribute to the evolution of ideas and knowledge in the topic of the study and the industrial practice component should inform decision making with regards to the choice of a methodology or technology that enables an organisation to benefit from the output of an academic study. SE research makes a theoretical contribution by typically following a pattern that gravitates from practice to theory (induction) and from theory to practice (deduction).

The methodology implemented for the current study comprised of an inductive approach that entailed the collection of data from a phenomenological perspective to propose a theoretical model that aligns OC to software development methodology. The study also implemented an element of abductive inference that leveraged the phenomenological data with the researcher's semantic interpretation of the data to propose a methodology for software development that integrates software development processes with operations processes (referred to as SDOM). The theoretical contribution that the current study makes to the domain of SE is represented as an illustration aligned to the Sjøberg *et al* model for a theoretical contributon to the domain of SE, illustrated in Figure 7.1.



Research

An *empirically* based study that has made a pioneering contribution by:

- developing a model that aligns specific agile development methods to organisational culture
- developing an agile based software process model that integrates software *development* processes with software *operations* processes paving the way for a trajectory of research that integrates software process models with operations processes
- proposing an adjustment to software development adoption behaviour theory to incorporate a dimension for organisational culture

Practice

An *empirically* based study that has made a pioneering contribution by:

- developing a model, structured according to organisational culture theory, that guides industry based transition initiatives on the adoption of agile methodology for software development, by illustrating a sequence that starts with the Waterfall approach, transitions to an intermediate Wagile approach and culminates in a fully fledged agile approach
- developing a Scrum based model that integrates software development processes with operations processing such as continuous integration, delivery and deployment thereby enabling the acquisition of better value from current software development process models

Figure 7.1: Theoretical Contribution of the Study

As illustrated in Figure 7.1, the study's main research contribution is conceptually placed in the domain of the OC and ASDM. The study's contribution is contextualised by drawing reference to 2 articles on the current state of agile software development practice. The first is a meta-study of the challenges and success factors of agile adoption by Dikert *et al.* (2016). The most influential factor that attenuates the adoption of agile methodology is a resistance to change that usually manifests in a top-down or hierarchical management style that incurs the prospect of "…reverting to an old way of working" (p. 97). In many instances, agile methodology degenerates into the 'management friendly' Waterfall methodology because of a lack of commitment by middle and senior management to embrace the changing values and practices espoused by agility. The study makes a call for research efforts that addresses the role played by organisational management in enabling the transformation to agility as well as the adaptation of agile methods so that these methods can scale to an organisational level. The second study is an article by Dingsøyr and Lassenius (2016) that entailed a compilation of research interests between the academic and practitioner communities over the last decade. An outcome of this comparison is the mismatch of research trends between these 2 groups. The academic sector has focused their research efforts on agile methods that are intrinsic to the 'inner workings' of XP and Scrum. The practitioner community has also bestowed a significant focus on Scrum, but more from the perspective of its use in an organisational/enterprise-wide context. The main source of inquiry from the practitioner community is to discover ways in which Scrum can be tailored so that it scales to an organisational platform. The methods that have been commonly touted as viable areas for future studies are continuous integration, continuous deployment and DevOps. There is a major focus on leveraging Scrum methodology to achieve business value by enabling a collaborative environment between software developers and operations/infrastructure engineers.

Based on the assumption that the 'sweetspot' for software engineering research has been identified by Dingsøyr and Lassenius (2016) and Dikert *et al.* (2016), then the current study has made a contribution by proposing 2 models that fit into the 'sweetspot' for software engineering research. Aligned to the Dikert *et al.* call for research that is grounded in the influence of management on the implementation of ASDM, the current study has leveraged empirical data to develop a framework that provides a guide on the influence of OC and management control on the implementation of ASDM. The study is also congruent with the Dingsøyr and Lassenius imperative to align academic research in SE to the requirements of industrial practice. This imperative has been achieved by the development of SDOM, a Scrum based software process model that comprises of elements of continuous integration, continuous deployment and the strategy of DevOps. SDOM has been tailored so that it is conveyed as a lightweight model thereby incurring an unobtrusive adjustment to the prevailing culture within an organisation. The objective of presenting such a lightweight model is that it does not have a disruptive influence on the prevailing OC. Models such as SDOM may be seen as a catalyst that will spawn the development of similar lightweight models by the academic research community, which can build upon the conceptualisation that software process models cannot exist in isolation from the operations activities required to galvanise these models to provide value to an organisation. The underlying philosophy behind such a stance is that it resonates with the principles of *simplicity* and *continuous delivery of working software* as espoused in the Agile Manifesto (see Fowler & Highsmith, 2001)

An additional contribution made by the study emanates from the structural equation modelling (SEM) exercise that was undertaken in relation to the theoretical model used to operationalise acceptance of SDOM. The theoretical model consists of the 4 independent variable constructs of PU, SN, CO and OS and the dependent variable construct of BI to use SDOM. The regression analysis and the SEM exercise reveals that the construct of CO did not make a significant contribution to the predictive capacity of the theoretical model and there was a high level of covariance between SN and OS. PU was however recognised as a strong predictor of BI to use SDOM. SEM was used to reconfigure the theoretical model so that those items used to measure SN, OS and CO that were perceived to be indicators of OC were conflated into the construct named OC. PU was retained as the other main construct of the model. The SEM exercise revealed that the overall predictive capacity of the newly configured theoretical model resulted in an improvement over the original model. This result may be subjected to scrutiny because the items used to measure OC did not have a strong theoretical underpinning. However, the conceptual outcome is significant in the sense that the SEM exercise provides an indicator that OC has to be included with PU as 2 of the main constructs in any model that is designed to operationalise the acceptance of a software development methodology.

7.5 Limitations of the Study

The Study's Scope and Sampling Strategy

The sampling strategy used in both the qualitative and quantitative phases of the study was purposive sampling which may be perceived as a limitation of the study from a generalisability perspective. However, as explained in in sections 3.2 and 3.2.5 of Chapter 3, the researcher has an interpretivist worldview orientation, thereby explaining the gravitation towards a predominantly qualitative approach for the first and defining phase of the study. The qualitative research approach resonates with the purposive sampling strategy adopted in the first phase of study where individuals who are experienced software practitioners were sampled using a phenomenological approach to generate 'rich' quality data on their experience of using a software development methodology. The phenomenological approach has yielded quality data, supplemented with input from 3 experienced and highly respected international practitioners in the SE domain. The contention made is that the ideas generated in the study have global applicability and a future study could seek empirical verification of these ideas on a broader, global platform.

The 2nd (quantitative) phase of the study entailed an acceptance study of one of the models that were output from the qualitative/exploratory phase of the study. A cohort of 40 purposively selected software practitioners provided a survey based acceptance response to the proposed SDOM. The use of purposive sampling in the quantitative phase of the study may also be perceived as a limitation of the study from a generalisability perspective. However, the exploratory nature of the study necessitated the use of a strategy whereby quality input could be obtained from subjects who are qualified to provide a meaningful response. The expansive testing of SDOM could be achieved in a subsequent study where there is an explicit focus on achieving external validity.

Completeness of the Validation Phase

The acceptance based quantitative phase of the study was directed on the technical aspect as represented by SDOM. However, an acceptance and validation exercise for the socially oriented OC model was not conducted because the validation of a model that guides ASDM on the basis of an amorphous concept such as OC can only be achieved if the respondents of the study have extensive familiarity with OC theory. The development of the OC model has been achieved on the basis of qualitative empirical data from experienced software practitioners coupled with the interpretive analysis of the researcher. Iivari and Huisman (2007) refer to such a model as an "empirically inspired theory" (p. 48), the type of which is severely lacking in information systems research.

Comprehensive Review of Agile Methodologies

The current study's review and analysis of agile methodologies has been confined to XP, Scrum and the Scrum/XP hybrid model. The rationale behind this strategy is that XP has been widely recognised as one of the pioneering agile methodologies (Abrahamsson *et al.*, 2017) and Scrum is currently the most widely used methodology for software development followed by the Scrum/XP hybrid model (VersionOne, 2016). The study's focus on Scrum and the Scrum/XP hybrid has been driven by the reported trends in software development methodology as well as the empirical data that converged to a viewpoint that Scrum has been endorsed as the de facto methodology for software development by South African organisations. It is recommended that an operations-oriented customisation of a broader range of agile methodologies be undertaken in a future study. The selection of agile methodologies for such a study may be made from the comprehensive review of agile methodologies provided in Abrahamsson *et al.* (2017).

Lack of Focus on Software Correctness and the CMMI-Dev

The current study's focus is to provide an extension to ASDM by incorporating elements of the operations domain into the software development process. This strategy will enhance the prospect for software systems to be delivered on *time* and within *budget*. The empirical data from the study suggests that the SDOM model achieves this objective. The researcher does however concede that SDOM does not have a specific focus on ensuring *software correctness*. While the testing phases within SDOM is oriented towards user acceptance testing, this is no guarantee that the system is correct. The lack of focus on system correctness is a limitation of the study. This limitation does however open up an opportunity for a subsequent study to integrate correctness testing as a specific phase in the agile development process. The advantage of integrating correctness testing into the process is that it enhances the prospect of aligning agile methodology to the Capability Maturity Model Integration for Development (CMMI-Dev). The CMMI-Dev is a model that provides organisations with a comparative framework to assess the level of maturity that they have reached in developing software. The lower levels of the CMMI-Dev model refers to software processes that are lacking in structure. The higher levels of the CMMI-Dev is a reference to processes that are repeatable where the focus is on defect control so that all new projects have lower instances of errors because of lessons learnt from previous systems development exercises.

7.6 Implications and Future Research

The study's implications will be contextualised by explaining 'what has been achieved' and 'what still needs to be achieved'.

Scrum Achieves Widespread Acceptance

The empirical evidence from the study attests to widespread acceptance of ASDM with a specific preference by South African organisations for a Scrum based approach to software development. Organisations are using hybridised versions of Scrum so that it aligns with organisational values and the requirements of the project being developed. The main form of hybridisation entails an integration of Waterfall practices such as BDUF and XP practices such as test driven development and continuous integration with core Scrum methods such as time boxing, the daily stand up meetings, the Scrum sprint and the sprint review. Scrum as a software process model is seen as a huge improvement over the Waterfall process model. There is however, a need to scale the Scrum-hybrid methodologies to the organisational platform so that there is a continuous delivery of business value. The current study has proposed a lightweight Scrum/DevOps based model that provides an integrated model of processes and role players who will be pivotal in enhancing the scalability of Scrum. The empirical evidence attests to substantial support for such a model where there is a conflation of the development and operations working environment. The lightweight nature of SDOM also facilitates easy acceptance because of a minimalistic deviation from the traditional Scrum model. The advantage of investing in such lightweight models is that it does not necessitate a drastic change in the prevailing OC. A further example of such a lightweight Scrum/DevOps model, where the focus is more on security and testing is proposed by Yasar and Kontostathis (2016), and discussed in Section 6.6 of Chapter 6 of the current study.

The viability of implementing lightweight models that have a Scrum/DevOps demeanour to obviate the 'disconnect' between development and operations needs to be further explored using a wider sample of software practitioners and organisations. The conceptual outcome of such studies would be that a trajectory of research in software development methodology is initiated, where the development methodology is integrated with operations processes that ultimately enable the delivery of business value at organisational level. Further studies are needed to explore how developers can contribute towards ensuring that the development code base is compatible with the operations code at the infrastructure level and a corollary to this would be studies that seek to determine how operations engineers can provide an operations platform that is compatible with the technologies that software developers are comfortable with using for development.

OC is recognised as a Predictor of ASDM Acceptance

OC plays a pivotal role in determining the acceptance of a software developmental methodology. Migration to an ASDM requires a shift of the OC in a direction that is less hierarchical. In many South African organisations, the culture of upper and middle management is hierarchically oriented. Changing such a 'deeply set' culture cannot be achieved in a short space of time. Incremental changes to the OC in a direction that is less hierarchical may be accompanied by incremental changes to the software development approach in a direction that is more agile. A framework that illustrates this transition has been provided in the current study. The study has also provided empirical evidence to support the assertion that the transition to ASDM may be accomplished incrementally by adopting lightweight models of agility such as SDOM thereby minimising the prospect of imposing a disruptive influence on the prevailing OC.

The academic research community should make further contributions in this regard by conducting empirical studies to ascertain the acceptance of software process models that have an agile orientation and provides direction for the attainment of business value. The theoretical models that are used to underpin such acceptance based studies should include socially oriented constructs that factor in the influence of OC. The socially oriented construct of Compatibility should also be operationalised by implementing the suggestion by Karahanna *et al.* (2006) of dissecting this construct into specific dimensions that measure compatibility with *organisational values* and compatibility with *personal values*.

The Scaled Agile Framework (SAFe) Intervention

Aligned to the imperative to improve the scalability of agile methods, the SAFe model is currently receiving substantial focus (VersionOne, 2016). From the study's data it has been established that two of the organisations from the banking sector in South Africa have made a commitment to adopt the SAFe model to achieve scalability of ASDM. However, there is a concession from the interviewee representatives from these organisations that SAFe is a 'disruptive' framework that requires comprehensive training and a firm organisational commitment towards a Developmental Culture orientation. From the empirical data, it has also been ascertained that a third organisation, also from the banking sector, does not see SAFe as a viable framework. The main reason for this negativity is the perception that SAFe conflicts with their preference for the current Group Culture orientation that leverages experiential knowledge to derive hybridised lightweight Scrum variants that have a better fit with the organisation's business processes. This kind of differentiation in the implementation of agile frameworks to achieve an optimal alignment with the OC provides a rich source of data for future comparative studies that could determine the acceptance of lightweight agile models compared to organisation-wide interventions such as SAFe.

7.7 Autobiographical Reflection

At the inception of the current study, the researcher's epistemological stance towards SE research has been conveyed in Ranjeeth *et al.* (2013). From an overview perspective this stance resonates with two main ideas. The first is that SE has to have a social and technical dimension that should not be dichotomous. The second is that academic research in SE needs to be pragmatically applicable in an industrial setting thereby dispelling the belief by software practitioners that academic research in SE has an 'ivory tower' orientation. The ideology emanating from this philosophical stance resulted in a research agenda that has the primary objective of making a contribution to the evolution of software development methodology and the secondary objective of paving a path that bridges the divide between academic research and its applicability in an industrial setting.

Achieving the objectives alluded to in the preceding paragraph have been challenging and rewarding. The main challenge faced by the researcher is the lack of academic literature on the strategies such as continuous integration and deployment to enable the scalability of agile methodology to an organisational platform. The plethora of academic studies on the acceptance and adoption of the 'inner workings' of agile methodology had somewhat of a misdirecting influence on the literature review and the preliminary design of the interview protocol used in the qualitative phase of the study. However, during the data collection phase, this shortcoming was soon realised and rectified. This adjustment resulted in a change in the orientation of the study from the 'inner workings' of ASDM to the role that it plays from an organisational and business value context. This realignment was crucial to arguably ensure that the study achieved its objective of making a contribution that is deemed to be relevant and applicable to an industrial setting.

A further challenge was to 'keep pace' with the rapid change of the capacity of technological tools to support the software development and deployment process.

The availability of technical support for strategies such as continuous integration and deployment, necessitated continual adjustment of the data collection plans so that the correct people were identified to arguably ensure that the study had a 'rich' source of data. The busy schedule of many of these software practitioners meant that the time that could be availed for the purpose of the current study was at a premium. The priority attached to work related commitments resulted in a few instances where members of the sample group of software practitioners requested for a postponement of the scheduled interview. These postponements contributed to a fragmented data collection phase that disrupted the continuity of the qualitative data analysis phase.

The rewarding aspect of the current study is that the researcher was provided with an opportunity to engage with members of the professional software development community and acquire knowledge of current software development practice as well as the latest tools that were used to support the development of professional software. This knowledge was pivotal in meeting the objectives of the current study and also enabling the researcher to make an input into the design of the academic SE curriculum at the researcher's organisation of employment in the tertiary education sector. The input made is in the domain of analysis and design and the priority attached to BDUF, the use of open source technologies to support continuous integration with tools such as Git and Github, the role played by a continuous integration server such as Jenkins and the use of containerisation tools such as Docker. The open source option to support continuous integration provides the flexibility of integrating solutions from both proprietary and open source platforms into an executable application that may be run on diverse platforms. The overriding message emerging from the trends in the professional sector is that the software development process requires methodological support that embraces flexibility and technological support that enables collaborative development with the objective of maintaining the evolving system in a 'ready to deploy' state.

7.8 Concluding Remarks

The current study was conceived as a plan to make a contribution to the software process improvement imperative that has been a focal area in the academic discipline of software engineering. The underlying objective was to make a contribution to the incremental expansion of the current body of knowledge in the domain of software development methodology. As alluded to in the study, current academic knowledge of software development methodology is centered on the agile approach for software development. This trend is reflective of a paradigmatic shift from a prescriptive approach to software development to one that espouses flexibility and the ability to respond to a dynamic environment where changing functional requirements are embraced as an integral feature of the software development process. The current study has embraced the concept of agility and set out to contribute to the trajectory of academic research that focuses on the enhancement of an agile based software development process model. The strategy used was to leverage experiential knowledge of ASDM from expert software practitioners to identify aspects of the methodology that could become the focus of a software process improvement initiative. An outcome of this empirical incursion into the experiential dimension of ASDM is that a transition to agility has to be aligned to a corresponding shift in the prevailing culture that exists in an organisation. This knowledge became a catalyst for an exploration of the influence of OC on the adoption of a software development methodology, resulting in the development of a model that guides the adoption of ASDM according to an OC classification. The empirical evidence also attests to the need for ASDM to integrate the non-functional elements of the operational environment with the software development process so that the transition from development to deployment is a seamless activity. The study has made a pioneering contribution in this regard by infusing elements of the operational environment with core software development activities intrinsic to the Scrum development process.

The study has succeeded in maintaining the trajectory of the current body of academic knowledge of software development methodology by incrementally extending this knowledge in the direction of OC and the operational environment in which software is implemented. Paradigmatically, these interventions resonate with the current impetus to impart agility and simplicity to the software development process. However, it also adds the imperative to accord cognisance to the social and business context in which most software systems function. The implication is that software process models have to incorporate phases that enable the integration of business and operational requirements so that these models may be perceived as useful in a professional, organisational context.

REFERENCES

Abbas, N., Gravell, A. M., & Wills, G. B. (2008). Historical roots of Agile methods: where did “Agile thinking” come from? *Agile Processes in Software Engineering and Extreme Programming* (pp. 94-103): Springer.

Abeyratne, K. S. (2014). *Analyzing Student Learning Outcomes In Programming Course Using Individual Study Vs. Pair Programming.* Dissertation, Graduate School, North Dakota State University, Unpublished.

Abrahamsson, P., Conboy, K., & Wang, X. (2009). “Lots done, more to do”: the current state of agile systems development research. *European Journal of Information Systems, 18*(4), 281-284.

Abrahamsson, P., & Koskela, J. (2004). *Extreme programming: A survey of empirical data from a controlled case study.* Paper presented at the Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Adams, D. A., Nelson, R. R., & Todd, P. A. (1992). Perceived usefulness, ease of use, and usage of information technology: A replication. *MIS quarterly*, 227-247.

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

Agarwal, R., & Prasad, J. (1998). The antecedents and consequents of user perceptions in information technology adoption. *Decision Support Systems, 22*(1), 15-29. doi: http://dx.doi.org/10.1016/S0167-9236(97)00006-7

Agresti, A. (2018). *Statistical Methods for the Social Sciences*. New York: Pearson.

Ahmad, M. O., Markkula, J., & Oivo, M. (2016). *Insights into the perceived benefits of Kanban in software companies: Practitioners’ views.* Paper presented at the International Conference on Agile Software Development.

Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. (2010). *Agile software development: Impact on productivity and quality.* Paper presented at the Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on.

Aiello, B. (2017). Interview on Agile LifeCycle Management and DevOps. In S. Ranjeeth (Ed.).

Aiello, B., & Sachs, L. (2016). *Agile Application Lifecycle Management: Using DevOps to Drive Process Improvement*: Addison-Wesley Professional.

Aitken, A., & Ilango, V. (2013). *A comparative analysis of traditional software engineering and agile software development.* Paper presented at the System Sciences (HICSS), 2013 46th Hawaii International Conference on.

Ajzen, I. (1985). *From intentions to actions: A theory of planned behavior*: Springer.

Ali, M. A. (2012). Survey on the state of Agile practices implementation in Pakistan. *International journal of Information and Communication Technology Research (IJICTR), 2*.

Allen, I. E., & Seaman, C. A. (2007). Likert scales and data analyses. *Quality progress, 40*(7), 64.

Allison, I. (2015). Towards an agile approach to Software Process Improvement: addressing the changing needs of software products. *Communications of the IIMA, 5*(1), 8.

Alshamrani, A., & Bahattab, A. (2015). A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. *International Journal of Computer Science Issues (IJCSI), 12*(1), 106.

Alsultanny, Y. A., & Wohaishi, A. M. (2009). Essential Characteristics of Software Model that Provide the Software Quality Assurance. *International Review on Computers & Software, 4*(5).

Ambler, S. W. (2001). The agile unified process, 1, from http://www.ambysoft.com/unifiedprocess/agileUP.html#Philosophies

Ambler, S. W., & Holitza, M. (2012). *Agile For Dummies*: John Wiley & Sons, Inc.

Ambler, S. W., & Lines, M. (2012). *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*: IBM Press.

Ambler, S. W., & Lines, M. (2013). Going Beyond Scrum: Disciplined Agile Delivery. *Disciplined Agile Consortium. White Paper Series*.

Ambler, S. W., & Lines, M. (2016). *The Disciplined Agile Process Decision Framework.* Paper presented at the International Conference on Software Quality.

Arisholm, E., Gallis, H., Dyba, T., & Sjoberg, D. I. (2007). Evaluating pair programming with respect to system complexity and programmer expertise. *Software Engineering, IEEE Transactions on, 33*(2), 65-86.

Arkin, B., Stender, S., & McGraw, G. (2005). Software penetration testing. *IEEE Security & Privacy, 3*(1), 84-87.

Aspray, W., Keil-Slawik, R., & Parnas, D. (1997). *The history of software engineering*: Citeseer.

Aveson, D., & Fitzgerald, G. (2006). Methodologies for developing information systems: A historical perspective *The Past and Future of Information Systems: 1976–2006 and Beyond* (pp. 27-38): Springer.

Bai, X., Zhang, H., & Huang, L. (2011). *Empirical Research in Software Process Modeling: A Systematic Literature Review.* Paper presented at the 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM),.

Bannerman, P. L., Hossain, E., & Jeffery, R. (2012). *Scrum practice mitigation of global software development coordination challenges: A distinctive advantage?* Paper presented at the System Science (HICSS), 2012 45th Hawaii International Conference on.

Bannink, S. (2014). *Challenges in the Transition from Waterfall to Scrum–a Case study at Port Base.* Paper presented at the 20th Twente Student Conference on Information Technology.

Bano, M., & Zowghi, D. (2013). *User involvement in software development and system success: a systematic literature review.* Paper presented at the Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering.

Basili, V. R. (1989). *Software development: A paradigm for the future.* Paper presented at the Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International.

Basili, V. R., Heidrich, J., Lindvall, M., Münch, J., Regardie, M., Rombach, D., Seaman, C., & Trendowicz, A. (2013). Linking software development and business strategy through measurement. *arXiv preprint arXiv:1311.6224.*

Basili, V. R., & Turner, A. J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering, 1*(4), 390-396.

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*: Addison-Wesley Professional.

Bazeley, P., & Jackson, K. (2013). *Qualitative data analysis with NVivo*: Sage Publications Limited.

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70-77.

Beck, K. (2000). *Extreme programming explained: embrace change*: Addison-Wesley Professional.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development Retrieved 15th June, 2010, from http://agilemanifesto.org/

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001). Manifesto for agile software development.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001). Manifesto for agile software development. *The Agile Alliance*

Benington, H. D. (1987). *Production of large computer programs.* Paper presented at the ICSE.

Bennett, K. H., & Rajlich, V. T. (2000). *Software maintenance and evolution: a roadmap.* Paper presented at the Proceedings of the Conference on the Future of Software Engineering.

Bertens, J. W. (1995). *The idea of the postmodern: A history*: Psychology Press.

Black, B. (2017). Interview on DevOps and Scrum Methodology. In S. Ranjeeth (Ed.).

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69.

Boehm, B. (2002). Software engineering is a value-based contact sport. *IEEE Software, 19*(5), 95-96.

Boehm, B. (2006). *A view of 20th and 21st century software engineering.* Paper presented at the Proceedings of the 28th international conference on Software engineering.

Boehm, B. (2011). Some future software engineering opportunities and challenges. *The future of software engineering*, 1-32.

Boehm, B., & Hansen, W. J. (2000). Spiral development: Experience, principles, and refinements: DTIC Document.

Boehm, B., & Turner, R. (2003). *Observations on balancing discipline and agility*. Paper presented at the Proceedings of the Agile Development Conference, 2003. ADC 2003.

Boehm, B., & Turner, R. (2003). People factors in software management: lessons from comparing agile and plan-driven methods. *CrossTalk: The Journal of Defense Software Engineering, 16*(12), 4-8.

Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *Computer*(6), 57-66.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Booch, G. (1986). Object-oriented development. *Software Engineering, IEEE Transactions on*(2), 211-221.

Booch, G. (2001). Developing the future. *Commun. ACM, 44*(3), 118-121. doi: 10.1145/365181.365234

Booch, G. (2006). *Object Oriented Analysis & Design with Application*: Pearson Education India.

Booch, G. (2012). Understanding Issues that Influence Agile Software Development. In S. Ranjeeth (Ed.).

Boone, H. N., & Boone, D. A. (2012). Analyzing likert data. *Journal of extension, 50*(2), 1-5.

Bradley, C. (2014). Resurrecting the Much-Maligned Scrum of Scrums Retrieved 13/4/2015, 2015, from http://blog.scrum.org/resurrecting-the-much-maligned-scrum-of-scrums/

Breivold, H. P., Sundmark, D., Wallin, P., & Larsson, S. (2010). *What does research say about agile and architecture?* Paper presented at the Software Engineering Advances (ICSEA), 2010 Fifth International Conference on.

Brhel, M., Meth, H., Maedche, A., & Werder, K. (2015). Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology, 61*, 163-181.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE computer, 20*(4), 10-19.

Brown, M. E. (2013). *Data-Driven Decision Making as a Tool to Improve Software Development Productivity.* Walden University.

Bryant, A. (2000). *It's engineering Jim… but not as we know it: software engineering—solution to the software crisis, or part of the problem?* Paper presented at the Proceedings of the 22nd international conference on Software engineering.

Byrne, B. M. (2010). *Structural Equation Modeling With AMOS, 2nd Edition*. New York: Routledge.

Cameron, K. S., & Quinn, R. E. (2011). *Diagnosing and changing organizational culture: Based on the competing values framework*: John Wiley & Sons.

Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring–A systematic literature review. *Journal of Systems and Software, 110*, 85-100.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. *European Journal of Information Systems, 18*(4), 332-343.

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *Software, IEEE, 25*(1), 60-67.

Capretz, L. F. (2003). A brief history of the object-oriented approach. *ACM SIGSOFT Software Engineering Notes, 28*(2), 6.

Causevic, A., Sundmark, D., & Punnekkat, S. (2011). *Factors limiting industrial adoption of test driven development: A systematic review.* Paper presented at the Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on.

Chan, F. K., & Thong, J. Y. (2009). Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems, 46*(4), 803-814.

Chan, F. K. Y., & Thong, J. Y. L. (2009). Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems, 46*(4), 803-814. doi: http://dx.doi.org/10.1016/j.dss.2008.11.009

Chan, Z. C., Fung, Y.-l., & Chien, W.-t. (2013). Bracketing in phenomenology: only undertaken in the data collection and analysis process? *The Qualitative Report, 18*(30), 1.

Chandra Misra, S., Kumar, V., & Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management, 27*(4), 451-474.

Chang, H. C. (2010). A new perspective on twitter hashtag use: diffusion of innovation theory. *Proceedings of the American Society for Information Science and Technology, 47*(1), 1-4.

Chaos. (2015). Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch.

Chau, P. Y., & Hu, P. J. (2002). Examining a model of information technology acceptance by individual professionals: An exploratory study. *Journal of management information systems, 18*(4), 191-229.

Cheung, G. W., & Rensvold, R. B. (2002). Evaluating goodness-of-fit indexes for testing measurement invariance. *Structural equation modeling, 9*(2), 233-255.

Cho, J. (2008). Issues and Challenges of agile software development with SCRUM. *Issues in Information Systems, 9*(2), 188-195.

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software, 81*(6), 961-971. doi: http://dx.doi.org/10.1016/j.jss.2007.08.020

Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology, 57*, 21-31.

Clarke, P., Mesquida, A.-L., Ekert, D., Ekstrom, J., Gornostaja, T., Jovanovic, M., Johansen, J., Mas, A., Messnarz, R., & Villar, B. N. (2016). An Investigation of Software Development Process Terminology *Software Process Improvement and Capability Determination* (pp. 351-361): Springer.

Cockburn, A. (1999). *Characterizing people as non-linear, first-order components in software development.* Paper presented at the International Conference on Software Engineering 2000.

Cockburn, A. (2002). Agile Software Development Joins the" Would-Be" Crowd. *Cutter IT Journal, 15*(1), 6-12.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer, 34*(11), 131-133. doi: 10.1109/2.963450

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in Computers, 62*, 1-66.

Cohn, M. (2004). Advantages of User Stories for Requirements Retrieved 10/10/2014, from http://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements

Cohn, M. (2004). *User stories applied: For agile software development*: Addison-Wesley Professional.

Cohn, M. (2006). ScrumMaster: Appointed or Team-Selected?   , from https://www.mountaingoatsoftware.com/articles/scrummaster

Cohn, M. (2012). Agile Succeeds Three Times More Often Than Waterfall Retrieved 12 July 2012, from http://www.mountaingoatsoftware.com/blog/agile-succeeds-three-times-more-often-than-waterfall

Conradi, R., Lindvall, M., & Seaman, C. (2000). *Success factors for software experience bases: what we need to learn from other disciplines.* Paper presented at the Proc. ICSE'2000 Workshop on Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research', Limerick, Ireland.

Cooper, H. M. (1988). Organizing knowledge syntheses: A taxonomy of literature reviews. *Knowledge, Technology & Policy, 1*(1), 104-126.

Corbin, J., & Strauss, A. (2014). *Basics of qualitative research: Techniques and procedures for developing grounded theory*: Sage publications.

Crawford, B., Barra, C. L. d. l., Soto, R., Misra, S., & Monfroy, E. (2013). Creative Thinking in eXtreme Programming.

Creswell, J. W. (2012). *Qualitative inquiry and research design: Choosing among five approaches*: Sage.

Creswell, J. W. (2012). *Qualitative inquiry and research design: Choosing among five approaches* (3rd ed.): Sage.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches*: Sage publications.

Creswell, J. W., Plano Clark, V. L., Gutmann, M. L., & Hanson, W. E. (2003). Advanced mixed methods research designs. *Handbook of mixed methods in social and behavioral research, 209*, 240.

Cunningham, W. (1993). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger, 4*(2), 29-30.

Cusumano, M. A., & Selby, R. W. (1997). How Microsoft builds software. *Communications of the ACM, 40*(6), 53-61.

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *Software Engineering, IEEE Transactions on, 14*(10), 1453-1461.

Davis, F. D. (1985). *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. Massachusetts Institute of Technology, Sloan School of Management.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, 319-340.

De Graff, J. (2007). The Competing Values Assessment Overview. Retrieved from http://competingvalues.com/competingvalues.com/wp-content/uploads/2009/09/CV-Overview-16-page.pdf

de O Melo, C., Santana, C., & Kon, F. (2012). *Developers motivation in agile teams.* Paper presented at the Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on.

DeLone, W. H., & McLean, E. R. (1992). Information Systems Success: The Quest for the Dependent Variable. [Article]. *Information Systems Research, 3*(1), 60-95.

Denison, D. R., & Spreitzer, G. M. (1991). Organizational culture and organizational development: A competing values approach. *Research in organizational change and development, 5*(1), 1-21.

DeSanctis, G., & Poole, M. S. (1994). Capturing the complexity in advanced technology use: Adaptive structuration theory. *Organization science, 5*(2), 121-147.

di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G., & Vlasenko, J. (2013). Pair Programming and Software Defects--A Large, Industrial Case Study. *Software Engineering, IEEE Transactions on, 39*(7), 930-953.

Dick, A. J., & Zarnett, B. (2002). Paired programming and personality traits. *Proceedings of XP*, 26-29.

Dijkstra, E. (1970). 7.4 Structured programming. *Software Engineering Techniques*, 65.

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software, 119*, 87-108.

Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology, 77*, 56-60.

Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology, 77*(Supplement C), 56-60. doi: https://doi.org/10.1016/j.infsof.2016.04.018

Dingsøyr, T., & Moe, N. B. (2014). Towards Principles of Large-Scale Agile Development *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation* (pp. 1-8): Springer.

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

Dowling, M. (2007). From Husserl to van Manen. A review of different phenomenological approaches. *International journal of nursing studies, 44*(1), 131-142.

Duggan, E. W. (2004). Silver Pellets for Improving Software Quality. [Article]. *Information Resources Management Journal, 17*(2), 1-21.

Duncan, O. D. (1966). Path analysis: Sociological examples. *American journal of Sociology, 72*(1), 1-16.

Dwivedi, Y. K., Rana, N. P., Chen, H., & Williams, M. D. (2011). A Meta-analysis of the Unified Theory of Acceptance and Use of Technology (UTAUT) *Governance and sustainability in information systems. Managing the transfer and diffusion of IT* (pp. 155-170): Springer.

Dybâ, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology, 50*(9), 833-859.

Dybâ, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9–10), 833-859. doi: http://dx.doi.org/10.1016/j.infsof.2008.01.006

Dybâ, T., & Dingsoyr, T. (2009). What do we know about agile software development? *Software, IEEE, 26*(5), 6-9.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology, 50*(9), 833-859.

Dyba, T., Moe, N. B., & Mikkelsen, E. M. (2004). *An empirical investigation on factors affecting software developer acceptance and utilization of electronic process guides.* Paper presented at the Software Metrics, 2004. Proceedings. 10th International Symposium on.

Edmunds, R., Thorpe, M., & Conole, G. (2012). Student attitudes towards and use of ICT in course study, work and social activity: A technology acceptance model approach. *British journal of educational technology, 43*(1), 71-84.

Eklund, U., Olsson, H. H., & Strøm, N. J. (2014). *Industrial challenges of scaling agile in mass-produced embedded systems.* Paper presented at the International Conference on Agile Software Development.

Englander, M. (2012). The interview: data collection in descriptive phenomenological human scientific research. *Journal of Phenomenological Psychology, 43*(1), 13-35.

Erasmus, E., Rothmann, S., & Van Eeden, C. (2015). A structural model of technology acceptance: original research. *SA Journal of Industrial Psychology, 41*(1), 1-12.

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management (JDM), 16*(4), 88-100.

Evans, C., Professor Raymond Hackney, D., Rauniar, R., Rawski, G., Yang, J., & Johnson, B. (2014). Technology acceptance model (TAM) and social media usage: an empirical study on Facebook. *Journal of Enterprise Information Management, 27*(1), 6-30.

Fenton, N., PFIEEGER, S. L., & Glass, R. L. (1997). Science and substance: A challenge to software engineers. *Applying Software Metrics, 46*, 6.

Fichman, R. G. (2000). The diffusion and assimilation of information technology innovations. *Framing the domains of IT management: Projecting the future through the past, 105127*.

Fishbein, M., & Ajzen, I. (1975). *Belief, attitude, intention and behavior: An introduction to theory and research*.

Fitzgerald, B. (1997). The use of systems development methodologies in practice: a field study. *Information Systems Journal, 7*(3), 201-212.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200-213.

Fitzgerald, B., & Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176-189.

Flora, H. K., Wang, X., & Chande, S. (2014). Adopting an agile approach for the development of mobile applications. *International Journal of Computer Applications, 94*(17), 43-50.

Foster, J. J., Barkus, E., & Yavorsky, C. (2005). *Understanding and Using Advanced Statistics: A practical guide for students*: Sage.

Fowler, M. (2000). The New Methodology Retrieved 6/10/2014, 2014, from http://www.martinfowler.com/articles/newMethodology.html#rationalUnifiedProcess

Fowler, M. (2001). The new methodology. *Wuhan University Journal of Natural Sciences, 6*(1-2), 12-24.

Fowler, M. (2005). The New Methodology Retrieved 12/13/2014, 2014, from http://www.martinfowler.com/articles/newMethodology.html

Fowler, M. (2006, 01 May 2006). Continuous Integration Retrieved 12/11/2016, 2016, from https://martinfowler.com/articles/continuousIntegration.html

Fowler, M. (2006). Using an agile software process with offshore development. *Capturado em http://martinfowler. com/articles/agileOffshore. html*.

Fowler, M. (2013). Extreme Programming Retrieved 11/12/2014, 2014, from http://martinfowler.com/bliki/ExtremeProgramming.html

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development, 9*(8), 28-35.

Fowler, M., West, D., Humble, J., Thomas, Badiceanu, & Kirk. (2014). A Retake on the Agile Manifesto.

Frank, U., Strecker, S., Fettke, P., vom Brocke, J., Becker, J., & Sinz, E. (2014). The research field "modeling business information systems". *Business & Information Systems Engineering, 6*(1), 39-43.

Frankel, J. (2017). Interview on the Adoption of the Devops Strategy at a Banking institution in South Africa. In S. Ranjeeth (Ed.).

Fuggetta, A. (2000). *Software process: a roadmap.* Paper presented at the Proceedings of the Conference on the Future of Software Engineering.

Fuggetta, A., & Di Nitto, E. (2014). *Software process.* Paper presented at the Proceedings of the on Future of Software Engineering.

Gallis, H., Arisholm, E., & Dyba, T. (2003). *An initial framework for research on pair programming.* Paper presented at the Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on.

Gallivan, M., & Srite, M. (2005). Information technology and culture: Identifying fragmentary and holistic perspectives of culture. *Information and Organization, 15*(4), 295-338.

Geer, D., & Harthorne, J. (2002). *Penetration testing: A duet.* Paper presented at the Computer Security Applications Conference, 2002. Proceedings. 18th Annual.

Gefen, D., Karahanna, E., & Straub, D. W. (2003). Trust and TAM in online shopping: An integrated model. *MIS quarterly, 27*(1), 51-90.

George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and software Technology, 46*(5), 337-342.

George, J. F., Batra, D., Valacich, J. S., & Hoffer, J. A. (2004). *Object-oriented systems analysis and design*: Pearson Prentice Hall.

Gershon, R. R., Stone, P. W., Bakken, S., & Larson, E. (2004). Measurement of organizational culture and climate in healthcare. *Journal of Nursing Administration, 34*(1), 33-40.

Ghasemi, A., & Zahediasl, S. (2012). Normality tests for statistical analysis: a guide for non-statisticians. *International journal of endocrinology and metabolism, 10*(2), 486.

Gill, M. J. (2014). The possibilities of phenomenology for organizational research. *Organizational Research Methods, 17*(2), 118-137.

Given, L. M. (2008). *The Sage encyclopedia of qualitative research methods*: Sage Publications.

Glass, R. L. (1994). The software-research crisis. *Software, IEEE, 11*(6), 42-47.

Glass, R. L. (2004). Matching methodology to problem domain. *Communications of the ACM, 47*(5), 19-21.

Glazer, H., Dalton, J., Anderson, D., Konrad, M., & Shrum, S. (2008). CMMI or Agile: Why Not Embrace Both! Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm

Glesne, C. (2015). *Becoming qualitative researchers: An introduction*: Pearson.

Gliem, J. A., & Gliem, R. R. (2003). *Calculating, interpreting, and reporting Cronbach's alpha reliability coefficient for Likert-type scales*.

Gorla, N., & Lin, S.-C. (2010). Determinants of software quality: A survey of information systems project managers. *Information and Software Technology, 52*(6), 602-610.

Grant, C., & Osanloo, A. (2014). Understanding, Selecting, and Integrating a Theoretical Framework in Dissertation Research: Creating the Blueprint for Your" House". *Administrative Issues Journal: Education, Practice, and Research, 4*(2), 12-26.

Gravetter, F. J., & Wallnau, L. B. (2014). *Essentials of Statistics for the Behavioral Sciences*.

Grinyer, A. (2007). Investigating adoption of agile software development methodologies in organisations *Agile Processes in Software Engineering and Extreme Programming* (pp. 163-164): Springer.

Griswold, W. G., & Opdyke, W. F. (2015). The Birth of Refactoring: A Retrospective on the Nature of High-Impact Software Engineering Research. *Software, IEEE, 32*(6), 30-38.

Groenewald, T. (2004). A phenomenological research design illustrated. *International journal of qualitative methods, 3*(1), 42-55.

Gualtieri, M. (2011). Agile Software Is A Cop-Out; Here's What's Next. Retrieved from http://blogs.forrester.com/mike_gualtieri/11-10-12-agile_software_is_a_cop_out_heres_whats_next

Guba, E. G., & Lincoln, Y. S. (1994). Competing paradigms in qualitative research. *Handbook of qualitative research, 2*(163-194), 105.

Habermas, J. (1970). Towards a theory of communicative competence. *Inquiry, 13*(1-4), 360-375.

Hannay, J. E., Arisholm, E., Engvik, H., & Sjoberg, D. I. (2010). Effects of personality on pair programming. *Software Engineering, IEEE Transactions on, 36*(1), 61-80.

Hardgrave, B. C., Davis, F. D., & Riemenschneider, C. K. (2003). Investigating determinants of software developers' intentions to follow methodologies. *Journal of Management Information Systems, 20*(1), 123-152.

Hardgrave, B. C., & Johnson, R. A. (2003). Toward an information systems development acceptance model: the case of object-oriented systems development. *Engineering Management, IEEE Transactions on, 50*(3), 322-336.

Hart, C. (1998). *Doing a literature review: Releasing the social science research imagination*: Sage.

Henderson-Sellers, B. (2006). *Method Engineering: Theory and Practice.* Paper presented at the ISTA.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*(9), 120-127.

Highsmith, J. A. (2002). *Agile software development ecosystems*: Addison-Wesley Professional.

Hoda, R., Noble, J., & Marshall, S. (2010). Agile undercover: when customers don't collaborate. *Agile Processes in Software Engineering and Extreme Programming*, 73-87.

Hofstede, G., Neuijen, B., Ohayv, D. D., & Sanders, G. (1990). Measuring organizational cultures: A qualitative and quantitative study across twenty cases. *Administrative science quarterly*, 286-316.

Hofstede, G. H., & Hofstede, G. (2001). *Culture's consequences*: SAGE Publications, Incorporated.

Hofstee, E. (2006). Constructing a good dissertation. *Johannesburg: EPE.*

Hooper, D., Coughlan, J., & Mullen, M. (2008). Structural equation modelling: Guidelines for determining model fit. *Articles*, 2.

Horward, C. (2015). Bimodal IT: Models for delivering scalable innovation in traditional enterprises, from https://c.ymcdn.com/sites/misaontario.site-ym.com/resource/resmgr/MCIO_2015/Presentations/Bimodal_IT_-_Gartner.pdf

Hoskin, T. (2012). *Parametric and nonparametric: Demystifying the terms.* Paper presented at the Mayo Clinic.

Howell, D., Windahl, C., & Seidel, R. (2010). A project contingency framework based on uncertainty and its consequences. [Article]. *International Journal of Project Management, 28*(3), 256-264. doi: 10.1016/j.ijproman.2009.06.002

Hu, L. t., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural equation modeling: a multidisciplinary journal, 6*(1), 1-55.

Hu, Q., Dinev, T., Hart, P., & Cooke, D. (2012). Managing employee compliance with information security policies: the critical role of top management and organizational culture. *Decision Sciences, 43*(4), 615-660.

Huang, A. (2008). *Similarity measures for text document clustering.* Paper presented at the Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand.

Huberman, A. M., Miles, M. B., & Saldana, J. (2013). *Qualitative data analysis: A methods sourcebook*: SAGE Publications, Incorporated.

Huck, S. W. (2012). *Reading statistics and research* (6th ed.). Boston: Pearson.

Humble, J. (2017). Lecturer Biography. *University of California, Berkely Staff Pages* Retrieved 15/07/2017, 2017, from https://www.ischool.berkeley.edu/people/jez-humble

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*: Pearson Education.

Hummel, M. (2014). *State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development.* Paper presented at the System Sciences (HICSS), 2014 47th Hawaii International Conference on.

IBM, U. P. (1998). Best practices for software development teams. *A Rational Software Corporation White Paper*.

Iivari, J., & Huisman, M. (2007). The relationship between organizational culture and the deployment of systems development methodologies. *Mis Quarterly*, 35-58.

Iivari, J., & Iivari, N. (2011). The relationship between organizational culture and the deployment of agile methods. *Information and Software Technology, 53*(5), 509-520. doi: http://dx.doi.org/10.1016/j.infsof.2010.10.008

Ingale, S., & Jadhav, S. (2012). *Comparative Study of Software Development Models.* Paper presented at the International conference on advances in computing &management.

Jackson, M. (1995). *The world and the machine.* Paper presented at the Software Engineering, 1995. ICSE 1995. 17th International Conference on.

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). *The unified software development process* (Vol. 1): Addison-Wesley Reading.

Jamieson, S. (2004). Likert scales: how to (ab) use them. *Medical education, 38*(12), 1217-1218.

Jemielniak, D. (2008). Software engineers or artists? Programmers' identity choices. *Tamara Journal of Critical Organisation Inquiry, 7*(1/2), 21.

Jensen, R. W. (1981). Tutorial Series 6 Structured Programming. *Computer, 14*(3), 31-48.

Jensen, R. W. (2014). *Improving software development productivity: Effective leadership and quantitative methods in software management*: Pearson Education.

Jobs, S. (1995, 05/06/2014). Quotes About the Future, from http://stevejobsdailyquote.com/steve-jobs-quotes-about-the-future/

Johnson, R. (1999). Applying the technology acceptance model to a systems development methodology. *AMCIS 1999 Proceedings*, 197.

Jung, H.-W., Kim, S.-G., & Chung, C.-S. (2004). Measuring software product quality: A survey of ISO/IEC 9126. *IEEE software*(5), 88-92.

Kafle, N. P. (2013). Hermeneutic phenomenological research method simplified. *Bodhi: An Interdisciplinary Journal, 5*(1), 181-200.

Kai-ming Au, A., & Enderwick, P. (2000). A cognitive model on attitude towards technology adoption. *Journal of Managerial Psychology, 15*(4), 266-282.

Kanellopoulos, Y., & Yu, Y. (2015). Guest editorial: Special section: Software quality and maintainability. *Software Quality Journal, 23*(1), 77-78.

Karahanna, E., Agarwal, R., & Angst, C. M. (2006). Reconceptualizing compatibility beliefs in technology acceptance research. *MIS quarterly*, 781-804.

Kaur, R., & Sengupta, J. (2013). Software Process Models and Analysis on Failure of Software Development Projects. *arXiv preprint arXiv:1306.1068*.

Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*.

Kerzazi, N., & Adams, B. (2016). *Who needs release and devops engineers, and why?* Paper presented at the Continuous Software Evolution and Delivery (CSED), IEEE/ACM International Workshop on.

Khramov, Y. (2006). *The cost of code quality*. Paper presented at the Agile Conference, 2006.

Kim, G. (2013). Top 11 Things You Need To Know About DevOps. *h ttp://itrevolution. com/pdf/Top11ThingsToKnowAboutDevOps. pdf*.

Kim, H.-Y. (2013). Statistical notes for clinical researchers: assessing normal distribution (2) using skewness and kurtosis. *Restorative dentistry & endodontics, 38*(1), 52-54.

Kim, M., Zimmermann, T., & Nagappan, N. (2012). *A field study of refactoring challenges and benefits*. Paper presented at the Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering.

King, W. R., & He, J. (2006). A meta-analysis of the technology acceptance model. *Information & management, 43*(6), 740-755.

Kirk, D., & MacDonell, S. G. (2014). *Investigating a conceptual construct for software context.* Paper presented at the Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., & Pohthong, A. (2017). Robust statistical methods for empirical software engineering. *Empirical Software Engineering, 22*(2), 579-630.

Kniberg, H., & Farhang, R. (2008). *Bootstrapping scrum and XP under crisis a story from the trenches.* Paper presented at the Agile, 2008. AGILE'08. Conference.

Knuth, D. E. (2007). *Computer programming as an art.* Paper presented at the ACM Turing award lectures.

Kong, S. (2007). *Agile software development methodology: effects on perceived software quality and the cultural context for organizational adoption.* Rutgers The State University of New Jersey - Newark. Retrieved from http://books.google.co.za/books?id=JsihLEw41gAC&printsec=frontcover#v=onepage&q&f=false

Kropp, M., & Meier, A. (2015). Agile Success Factors.

Kropp, M., & Meier, A. (2015). Agile Success Factors. *Retrieved May, 12,* 2015.

Kruchten, P. (2004). Scaling down large projects to meet the agile “sweet spot”.

Kruchten, P. (2004). Scaling down large projects to meet the agile “sweet spot”. *IBM developerWorks, 13*.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software, 29*(6), 18-21.

Kuhn, T. S. (1970). BOOK AND FILM REVIEWS: Revolutionary View of the History of Science: The Structure of Scientific Revolutions. *The Physics Teacher, 8*(2), 96-98.

Kujala, S. (2003). User involvement: a review of the benefits and challenges. *Behaviour & information technology, 22*(1), 1-16.

Kundalram, V. (2013). *Requirements Elicitation as a Predictor of Information Systems Success.* KwaZulu-Natal.

Lal, R. (2011). *Strategic Factors In Agile Software Development Method Adaptation: A Study Of Market-Driven Organisations:*. PhD in Information Technology, Massey University, New Zealand.

Lalsing, V., Kishnah, S., & Pudaruth, S. (2012). People factors in agile software development and project management. *International Journal of Software Engineering & Applications (IJSEA), 3*(1), 117-137.

Lambrechts, F. J., Bouwen, R., Grieten, S., Huybrechts, J. P., & Schein, E. H. (2011). Learning to help through humble inquiry and implications for management research, practice, and education: An interview with Edgar H. Schein. *Academy of Management Learning & Education, 10*(1), 131-147. doi: 10.5465/AMLE.2011.59513279

Langdridge, D. (2008). Phenomenology and critical social psychology: Directions and debates in theory and research. *Social and Personality Psychology Compass, 2*(3), 1126-1142.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer, 36*(6), 47-56.

Larsen, D., Derby, E., Eckstein, J., Joiner, B., Viliki, K., Shore, J., Hamman, M., & Schlitz, G. (2012). Characteristics of Agile Organizations.

Laverty, S. M. (2003). Hermeneutic Phenomenology and Phenomenology: A Comparison of Historical and Methodological Considerations. *International Journal of Qualitative Methods, 2*(3), 21-35. doi: doi:10.1177/160940690300200303

Layman, L., Williams, L., & Cunningham, L. (2004). *Exploring extreme programming in context: An industrial case study*.

Lee, J.-C., Shiue, Y.-C., & Chen, C.-Y. (2016). Examining the impacts of organizational culture and top management support of knowledge sharing on the success of software process improvement. *Computers in Human Behavior, 54*, 462-474.

Lee, Y.-H., Hsieh, Y.-C., & Hsu, C.-N. (2011). Adding innovation diffusion theory to the technology acceptance model: Supporting employees' intentions to use e-learning systems. *Journal of Educational Technology & Society, 14*(4), 124.

Leedy, P. D., & Ormrod, J. E. (2005). Practical research: Planning and design.

Leffingwell, D. (2007). *Scaling software agility: best practices for large enterprises*: Pearson Education.

Leffingwell, D. (2010). *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*: Addison-Wesley Professional.

Legris, P., Ingham, J., & Collerette, P. (2003). Why do people use information technology? A critical review of the technology acceptance model. *Information & management, 40*(3), 191-204.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE, 68*(9), 1060-1076.

Lehman, M. M., & Ramil, J. F. (2003). Software evolution—background, theory, practice. *Information Processing Letters, 88*(1), 33-44.

Leidner, D. E., & Kayworth, T. (2006). Review: A review of culture in information systems research: Toward a theory of information technology culture conflict. *MIS quarterly, 30*(2), 357-399.

LeVasseur, J. J. (2003). The problem of bracketing in phenomenology. *Qualitative health research, 13*(3), 408-420.

Levy, Y., & Ellis, T. J. (2006). A systems approach to conduct an effective literature review in support of information systems research. *Informing Science: International Journal of an Emerging Transdiscipline, 9*, 181-212.

Li, J., Moe, N. B., & Dybå, T. (2010). *Transition from a plan-driven process to scrum: a longitudinal case study on software quality.* Paper presented at the Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement.

Limoncelli, T. A., & Hughes, D. (2011). LISA'11 Theme—"DevOps: New Challenges, Proven Values". *USENIX; login: Magazine, 36*(4).

Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information Systems Management, 21*(3), 41-52.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., & Zelkowitz, M. (2002). Empirical findings in agile methods. *Extreme Programming and Agile Methods—XP/Agile Universe 2002*, 81-92.

Lonchamp, J. (1993). *A structured conceptual and terminological framework for software process engineering.* Paper presented at the Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the.

Lott, J. P. (2015). A note on normality. *Journal of the American Academy of Dermatology, 72*(6), e169-e170.

Lovelace, M., & Brickman, P. (2013). Best practices for measuring students' attitudes toward learning science. *CBE-Life Sciences Education, 12*(4), 606-617.

Lyytinen, K., & Damsgaard, J. (2001). What's wrong with the diffusion of innovation theory. *Diffusing Software Products and Process Innovations*, 173-190.

Lyytinen, K., & Rose, G. M. (2006). Information system development agility as organizational learning. *European Journal of Information Systems, 15*(2), 183-199.

Machado, T. C. S., Pinheiro, P. R., & Tamanini, I. (2015). Project management aided by verbal decision analysis approaches: a case study for the selection of the best SCRUM practices. *International Transactions in Operational Research, 22*(2), 287-312.

Maciaszek, L. (2007). *Requirements analysis and system design*: Pearson Education.

Mahnič, V. (2008). Teaching Information System Technology in Partnership with IT Companies. *Organizacija, 41*(2), 71-78.

Mahoney, M. S. (2004). Finding a history for software engineering. *Annals of the History of Computing, IEEE, 26*(1), 8-19.

Malone, M. W. (2014). *Process subversion in Agile Scrum software development: A phenomenological approach.* Capella University.

Mann, C., & Maurer, F. (2005). *A case study on the impact of scrum on overtime and customer satisfaction.* Paper presented at the null.

Mannaro, K., Melis, M., & Marchesi, M. (2004). Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies *Extreme Programming and Agile Processes in Software Engineering* (pp. 166-174): Springer.

Marakas, G. M. (2006). *Systems analysis & design*: McGraw-Hill Irwin.

Martin, A., Biddle, R., & Noble, J. (2004). *The XP customer role in practice: Three studies.* Paper presented at the Agile Development Conference, 2004.

Mason, M. (2010). *Sample size and saturation in PhD studies using qualitative interviews.* Paper presented at the Forum qualitative Sozialforschung/Forum: qualitative social research.

Matthews, C. R. (2014). Software Quality Strategy Supported by People and Organizations.

Mayfield, K. M. (2010). *Project managers' experience and description of decision uncertainty associated with the agile software development methodology: A phenomenological study.* Capella University.

McAvoy, J., & Butler, T. (2009). The role of project management in ineffective decision making within Agile software development projects. *European Journal of Information Systems, 18*(4), 372-383.

McBreen, P. (2000). *Applying the Lessons of eXtreme Programming.* Paper presented at the TOOLS (34).

McClave, J. T. S., McClave, T. J. T., & Sincich, T. (2012). *Statistics*.

McConnell, S. (1996). Daily build and smoke test. *IEEE software, 13*(4), 144.

McCormick, M. (2001). Technical opinion: Programming extremism. *Communications of the ACM, 44*(6), 109-119.

McCracken, D. D., & Jackson, M. A. (1982). Life cycle concept considered harmful. *ACM SIGSOFT Software Engineering Notes, 7*(2), 29-32.

McHugh, M., McCaffery, F., & Casey, V. (2012). Barriers to adopting agile practices when developing medical device software *Software Process Improvement and Capability Determination* (pp. 141-147): Springer.

McLeod, L., & MacDonell, S. G. (2011). Factors that affect software systems development project outcomes: A survey of research. *ACM Computing Surveys (CSUR), 43*(4), 24.

Mehra, B. (2002). Bias in qualitative research: Voices from an online classroom. *The Qualitative Report, 7*(1), 1-19.

Melo, C. d. O., Ferraz, R., & Parsons, R. J. (2016). Brazil and the Emerging Future of Software Engineering. *Software, IEEE, 33*(1), 45-47.

Meso, P., & Jain, R. (2006). Agile software development: adaptive systems principles and best practices. *Information Systems Management, 23*(3), 19-30.

Meyer, B. (2003). *The grand challenge of trusted components.* Paper presented at the Software Engineering, 2003. Proceedings. 25th International Conference on.

Meyer, B. (2014). *Agile!: The Good, the Hype and the Ugly*: Springer Science & Business Media.

Mills, H. D. (1980). The management of software engineering, Part I: Principles of software engineering. *IBM Systems Journal*, *19*(4), 414-420.

Mingers, J. (2001). Combining IS research methods: towards a pluralist methodology. *Information systems research*, *12*(3), 240-259.

Minkov, M., & Hofstede, G. (2011). The evolution of Hofstede's doctrine. *Cross Cultural Management: An International Journal*, *18*(1), 10-20.

Miranda, M., Ferreira, R., de Souza, C. R., Figueira Filho, F., & Singer, L. (2014). *An exploratory study of the adoption of mobile development platforms by software engineers.* Paper presented at the Proceedings of the 1st International Conference on Mobile Software Engineering and Systems.

Mishra, D., & Mishra, A. (2011). Complex software project development: agile methods adoption. *Journal of Software Maintenance and Evolution: Research and Practice, 23*(8), 549-564.

Misra, S., Kumar, V., Kumar, U., Fantazy, K., & Akhter, M. (2012). Agile software development practices: evolution, principles, and criticisms. *International Journal of Quality & Reliability Management, 29*(9), 972-980.

Misra, S. C., Kumar, V., & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software, 82*(11), 1869-1890. doi: http://dx.doi.org/10.1016/j.jss.2009.05.052

Mnkandla, E. (2008). *A Selection Framework For Agile Methodology Practices: A Family of Methodologies Approach.* Faculty of Engineering and the Built Environment, University of The Witwatersrand.

Moniruzzaman, A., & Hossain, D. S. A. (2013). Comparative study on agile software development methodologies. *arXiv preprint arXiv:1307.3356*.

Montoni, M. A., & da Rocha, A. R. C. (2013). Applying grounded theory to understand software process improvement implementation: a study of Brazilian software organizations. *Innovations in Systems and Software Engineering*, 1-8.

Morandini, M., Penserini, L., Perini, A., & Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engineering, 22*(1), 77-103.

Morgan, D. L. (2007). Paradigms lost and pragmatism regained methodological implications of combining qualitative and quantitative methods. *Journal of mixed methods research*, *1*(1), 48-76.

Morse, J., & Richards, L. (2002). Read me first for a user's guide to qualitative research. *CA, US: Sage Publications Thousand Oaks*.

Morse, J. M. (1991). Approaches to qualitative-quantitative methodological triangulation. *Nursing research, 40*(2), 120-123.

Moustakas, C. (1994). *Phenomenological research methods*: Sage.

Mueller, E. (2016, 2016). What Is DevOps? Retrieved 12 January, 2017, from https://theagileadmin.com/what-is-devops/

Mundra, A., Misra, S., & Dhawale, C. (2013). *Practical Scrum-Scrum team: Way to produce successful and quality software.* Paper presented at the Computational Science and Its Applications (ICCSA), 2013 13th International Conference on.

Murphy, B., Bird, C., Zimmermann, T., Williams, L., Nagappan, N., & Begel, A. (2013). *Have Agile Techniques been the Silver Bullet for Software Development at Microsoft?* Paper presented at the Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on.

Mushtaq, Z., & Qureshi, M. R. J. (2012). Novel Hybrid Model: Integrating Scrum and XP. *International Journal of Information Technology and Computer Science (IJITCS), 4*(6), 39.

Mustonen-Ollila, E., & Lyytinen, K. (2003). Why organizations adopt information system process innovations: a longitudinal study using Diffusion of Innovation theory. *Information Systems Journal, 13*(3), 275-297.

Myers. (1997). Qualitative research in information systems. *Management Information Systems Quarterly, 21*(2), 241-242.

Nandhakumar, J., & Avison, D. E. (1999). The fiction of methodological development: a field study of information systems development. *Information technology & people, 12*(2), 176-191.

Naumann, J. D., & Jenkins, A. M. (1982). Prototyping: The New Paradigm for Systems Development. [Article]. *MIS Quarterly, 6*(3), 29-44.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 72-78.

Nerur, S., Mahapatra, R. K., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 72-78.

Nguyen, D. S. (2016). Workplace Factors that Shape Agile Software Development Team Project Success. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS), 17*(1), 323-391.

Ngwenyama, O., & Nielsen, P. A. (2003). Competing values in software process improvement: an assumption analysis of CMM from an organizational culture perspective. *Engineering Management, IEEE Transactions on, 50*(1), 100-112.

Niwattanakul, S., Singthongchai, J., Naenudorn, E., & Wanapu, S. (2013). *Using of Jaccard coefficient for keywords similarity.* Paper presented at the Proceedings of the International MultiConference of Engineers and Computer Scientists.

Nkukwana, S., & Terblanche, N. H. (2017). Between a rock and a hard place: management and implementation teams' expectations of project managers in an agile information systems delivery environment. *South African Journal of Information Management, 19*(1), 1-10.

Norman, G. (2010). Likert scales, levels of measurement and the "laws" of statistics. *Advances in health sciences education, 15*(5), 625-632.

Northover, M., Northover, A., Gruner, S., Kourie, D. G., & Boake, A. (2007). *Agile software development: a contemporary philosophical perspective*. Paper presented at the Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Port Elizabeth, South Africa.

Paasivaara, M., Lassenius, C., & Heikkila, V. T. (2012). *Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?* Paper presented at the Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on.

Parnas, D., & Clements, P. (1985). A rational design process: How and why to fake it. *Formal Methods and Software Development*, 80-100.

Patton, M. Q. (1990). *Qualitative evaluation and research methods*: SAGE Publications, inc.

Paulk, M. (2014). On Empirical Research Into Scrum Adoption: Carnegie Mellon University, viewed.

Paulk, M. C. (2001). Extreme programming from a CMM perspective. *Software, IEEE, 18*(6), 19-26.

Penny, S. (1997). The virtualization of art practice: Body knowledge and the engineering worldview. *Art Journal, 56*(3), 30-38.

Petersen, K., & Gencel, C. (2013). *Worldviews, research methods, and their relationship to validity in empirical software engineering research.* Paper presented at the Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on.

Petre, M. (2013). *UML in practice.* Paper presented at the Proceedings of the 2013 International Conference on Software Engineering.

Petter, S. C., & Gallivan, M. J. (2004). *Toward a framework for classifying and guiding mixed method research in information systems.* Paper presented at the System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on.

Pfleeger, S. L. (1999). Understanding and improving technology transfer in software engineering. *Journal of Systems and Software, 47*(2), 111-124.

Pfleeger, S. L., & Kitchenham, B. A. (2001). Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes, 26*(6), 16-18.

Pinch, T. (2008). Technology and institutions: living in a material world. *Theory and society, 37*(5), 461-483.

Pope, C., & Mays, N. (1995). Reaching the parts other methods cannot reach: an introduction to qualitative methods in health and health services research. *BMJ: British Medical Journal, 311*(6996), 42.

Port, D., & Bui, T. (2009). Simulating mixed agile and plan-based requirements prioritization strategies: proof-of-concept and practical implications. *European Journal of Information Systems, 18*(4), 317-331.

Pozzebon, M., Mackrell, D., & Nielsen, S. (2014). Structuration bridging diffusion of innovations and gender relations theories: a case of paradigmatic pluralism in IS research. *Information Systems Journal*, *24*(3), 229-248.

Pressman, R. S. (2010). *Software engineering: a practitioner's approach*: McGraw-Hill Higher Education.

Quinn, R. E., & McGrath, M. R. (1985). The transformation of organizational cultures: A competing values perspective. *Organizational culture*, 315-334.

Quinn, R. E., & McGrath, M. R. (1985). The transformation of organizational cultures: A competing values perspective.

Radermacher, A. D., & Walia, G. S. (2011). *Investigating the effective implementation of pair programming: an empirical investigation.* Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education.

Ranasinghe, R. C., & Perera, I. (2015). *Effectiveness of scrum for offshore software development in Sri Lanka.* Paper presented at the Moratuwa Engineering Research Conference (MERCon), 2015.

Ranjeeth, S., Marimuthu, M., & Maharaj, M. (2013). A Pedagogical Intervention Based on Agile Software Development Methodology. *Alternation Special EditionTrends in Management, Informatics and Research in a 21st Century Digitally Connected World., 8*(2013), 225-250.

Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). DevOps in the Ascendency *DevOps for Digital Leaders* (pp. 3-14): Springer.

Reid Turner, C., Fuggetta, A., Lavazza, L., & Wolf, A. L. (1999). A conceptual basis for feature engineering. *Journal of Systems and Software, 49*(1), 3-15.

Remler, D. K., & Van Ryzin, G. G. (2011). *Research methods in practice: Strategies for description and causation*: Sage Publications.

Riemenschneider, C., & Hardgrave, B. (2001). Explaining software development tool use with the technology acceptance model. *Journal of Computer Information Systems, 41*(4), 1-8.

Riemenschneider, C., Hardgrave, B., & Davis, F. (2002). Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *Software Engineering, IEEE Transactions on, 28*(12), 1135-1145.

Riemenschneider, C. K., Hardgrave, B. C., & Davis, F. D. (2002). Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *IEEE transactions on Software Engineering, 28*(12), 1135-1145.

Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE software, 17*(4), 26-32.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). *DevOps Adoption Benefits and Challenges in Practice: A Case Study.* Paper presented at the Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17.

Robinson, H., Hall, P., Hovenden, F., & Rachel, J. (1998). Postmodern software development. *The Computer Journal, 41*(6), 363-375.

Robinson, H., & Sharp, H. (2009). The emergence of object-oriented technology: the role of community. [Article]. *Behaviour & Information Technology, 28*(3), 211-222. doi: 10.1080/01449290701494548

Roche, J. (2013). Adopting DevOps practices in quality assurance. *Communications of the ACM, 56*(11), 38-43.

Rogers, E. M. (1983). *Diffusion of innovations*. New York: Free Press.

Rowley, J. (2014). Designing and using research questionnaires. *Management Research Review, 37*(3), 308-330.

Royce, W. W. (1970). *Managing the development of large software systems.* Paper presented at the proceedings of IEEE, Los Angeles.

Royce, W. W. (1970). *Managing the development of large software systems.* Paper presented at the proceedings of IEEE WESCON.

Rubin, H., & Rubin, I. (2005). The art of qualitative interviewing: Thousand Oaks, CA: SAGE.

Rubin, H. J., & Rubin, I. S. (2012). *Qualitative interviewing: The Art of Hearing Data*: Sage.

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*: Addison-Wesley.

Rumpe, B., & Schröder, A. (2014). Quantitative survey on extreme programming projects. *arXiv preprint arXiv:1409.6599*.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes, 35*(3), 8-13.

Ryan, K. T. (2015). Software processes for a changing world. *Journal of Software: Evolution and Process*.

Ryan, S., & O'Connor, R. V. (2013). Acquiring and Sharing Tacit Knowledge in Software Development Teams: An Empirical Study. *Information and Software Technology*.

Saadé, R., & Bahli, B. (2005). The impact of cognitive absorption on perceived usefulness and perceived ease of use in on-line learning: an extension of the technology acceptance model. *Information & management, 42*(2), 317-327.

Sahota, M. (2012). *An Agile Adoption And Transformation Survival Guide*. http://www.infoq.com/minibooks/agile-adoption-transformation: Infotrix.

Saldana, J. (2009). An introduction to codes and coding. *The coding manual for qualitative researchers*, 1-31.

Saldaña, J. (2015). *The coding manual for qualitative researchers*: Sage.

Saunders, M. N. (2011). *Research methods for business students, 5/e*: Pearson Education India.

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., & Stumm, M. (2016). *Continuous deployment at Facebook and OANDA.* Paper presented at the Proceedings of the 38th International Conference on Software Engineering Companion.

Scacchi, W. (1987). Models of software evolution: life cycle and process: DTIC Document.

Scaled_Agile. (2017). *SAFe 4.5 Introduction*. Retrieved from https://www.scaledagile.com/resources/safe-whitepaper/

Schach, S. R. (2008). *Object-oriented software engineering*: McGraw-Hill.

Scheerer, A., Hildenbrand, T., & Kude, T. (2014). *Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective.* Paper presented at the System Sciences (HICSS), 2014 47th Hawaii International Conference on.

Schein, E. H. (1983). The Role of the Founder in the Creation of Organizational Culture: DTIC Document.

Schein, E. H. (1985). Defining organizational culture. *Classics of organization theory, 3*, 490-502.

Schein, E. H. (1996). Leadership and organizational culture. *The leader of the future, 11*, 116.

Schepers, J., & Wetzels, M. (2007). A meta-analysis of the technology acceptance model: Investigating subjective norm and moderation effects. *Information & Management*, *44*(1), 90-103.

Schwaber, K. (1997). Scrum development process *Business Object Design and Implementation* (pp. 117-134): Springer.

Scotland, J. (2012). Exploring the philosophical underpinnings of research: Relating ontology and epistemology to the methodology and methods of the scientific, interpretive, and critical research paradigms. *English Language Teaching, 5*(9), 9.

Sekaran, U., & Bougie, R. (2010). *Research methods for business : a skill-building approach* (5th ed. ed.). Hoboken, N.J.: Wiley ; Chichester : John Wiley [distributor].

Senapathi, M., & Srinivasan, A. (2012). Understanding post-adoptive agile usage: an exploratory cross-case analysis. *Journal of Systems and Software, 85*(6), 1255-1268.

Serrador, P., & Pinto, J. K. (2015). Does Agile work?—A quantitative analysis of agile project success. *International Journal of Project Management, 33*(5), 1040-1051.

Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2017). Adopting Continuous Delivery and Deployment: Impacts on Team Structures, Collaboration and Responsibilities.

Sharma, S. (2017). Adopting DevOps *The DevOps Adoption Playbook* (pp. 39-65): John Wiley & Sons, Inc.

Sharma, S. (2017). DevOps: An Overview *The DevOps Adoption Playbook* (pp. 1-38): John Wiley & Sons, Inc.

Sharp, H., & Robinson, H. (2005). Some social factors of software engineering: the maverick, community and technical practices. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-6. doi: 10.1145/1082983.1083117

Sheffield, J., & Lemétayer, J. (2013). Factors associated with the software development agility of successful projects. *International Journal of Project Management, 31*(3), 459-472.

Sidky, A., Arthur, J., & Bohner, S. (2007). A disciplined approach to adopting agile practices: the agile adoption framework. *Innovations in systems and software engineering, 3*(3), 203-216.

Simberova, I. (2015). Corporate culture–as a barrier of market orientation implementation. *Economics and Management*(14), 513-521.

Sinha, A., & Prajapati, D. (2014). Review on scrum and extreme programming for software quality assurance in industries. *International Journal of Management, IT and Engineering, 4*(10), 352-362.

Sjøberg, D. I., Dybå, T., Anda, B. C., & Hannay, J. E. (2008). Building theories in software engineering. *Guide to advanced empirical software engineering*, 312-336.

Slaten, K. M., Droujkova, M., Berenson, S. B., Williams, L., & Layman, L. (2005). *Undergraduate student perceptions of pair programming and agile software methodologies: Verifying a model of social interaction*.

Smith, J. A., Flowers, P., & Osborn, M. (1997). Interpretative phenomenological analysis and the psychology of health and illness. *Material discourses of health and illness*, 68-91.

Söderland, J., Geraldi, J., Müller, R., & Jugdev, K. (2012). Critical success factors in projects: Pinto, Slevin, and Prescott-the elucidation of project success. *International Journal of Managing Projects in Business, 5*(4), 757-775.

Sommerville, I. (1996). Software process models. *ACM Computing Surveys (CSUR), 28*(1), 269-271.

Sommerville, I. (2007). *Software Engineering*: Addison-Wesley.

Steinskog, D. J., Tjøstheim, D. B., & Kvamstø, N. G. (2007). A cautionary note on the use of the Kolmogorov–Smirnov test for normality. *Monthly Weather Review, 135*(3), 1151-1157.

Stern, R. (2002). *Routledge philosophy guidebook to Hegel and the phenomenology of spirit*: Psychology Press.

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software development: Agile vs. traditional. *Informatica Economica, 17*(4), 64.

Stol, K.-J., Ralph, P., & Fitzgerald, B. (2016). *Grounded theory in software engineering research: a critical review and guidelines.* Paper presented at the Proceedings of the 38th International Conference on Software Engineering.

Strauss, A., & Corbin, J. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory*: Sage Publications, Inc.

Strode, D. E. (2012). A Theory of Coordination in Agile Software Development Projects.

Strode, D. E., Huff, S. L., & Tretiakov, A. (2009). *The impact of organizational culture on agile method use.* Paper presented at the System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on.

Sutherland, J. (2001). Agile can scale: Inventing and reinventing scrum in five companies. *Cutter IT journal, 14*(12), 5-11.

Sutherland, J., & Schwaber, K. (2009). Scrum Guide Retrieved 10/03/2015, 2015, from http://www.scrumalliance.org/resources>,

Sutherland, J., Schwaber, K., Scrum, C.-c. O., & Sutherl, C. J. (2012). The scrum papers: Nuts, bolts, and origins of an agile process Retrieved 11/11, 2014, from http://jeffsutherland.com/ScrumPapers.pdf

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard business review, 64*(1), 137-146.

Tanner, M., & Seymour, L. (2014). *The range and level of software development skills needed in the Western Cape, South Africa.* Paper presented at the Proceedings of the e-skills for knowledge production and innovation conference 2014, cape town, south africa.

Templeton, G. F., & Byrd, T. A. (2003). Determinants of the relative advantage of a structured SDM during the adoption stage of implementation. *Information Technology and Management, 4*(4), 409-428.

Tessem, B., & Maurer, F. (2007). Job satisfaction and motivation in a large agile team *Agile Processes in Software Engineering and Extreme Programming* (pp. 54-61): Springer.

Theocharis, G., Kuhrmann, M., Münch, J., & Diebold, P. (2015). *Is water-scrum-fall reality? on the use of agile and traditional development practices.* Paper presented at the International Conference on Product-Focused Software Process Improvement.

Thomas, D. (2015). Agile is Dead. Amsterdam.

Thomas, D. R. (2006). A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation, 27*(2), 237-246.

Tolfo, C., Wazlawick, R. S., Ferreira, M. G. G., & Forcellini, F. A. (2011). Agile methods and organizational culture: Reflections about cultural levels. *Journal of Software Maintenance and Evolution: Research and Practice, 23*(6), 423-441.

Tomayko, J. E. (2002). A comparison of pair programming to inspections for software defect reduction. *Computer Science Education, 12*(3), 213-222.

Trauring, A. (2002). Software methodologies: The battle of the Gurus. *Info-Tech White Papers*.

Treacy, F., Rooney, M., Slattery, S., Staunton, C., & McHugh, O. (2008). *A Study of XP & SCRUM: A Project Management Perspective.* Paper presented at the Proceedings of 3rd International Business Informatics Challenge and Conference.

Tripp, J. F., & Riemenschneider, C. K. (2014). *Toward an Understanding of Job Satisfaction on Agile Teams: Agile Development as Work Redesign.* Paper presented at the System Sciences (HICSS), 2014 47th Hawaii International Conference on.

Turk, D., France, R., & Rumpe, B. (2014). Assumptions underlying agile software development processes. *arXiv preprint arXiv:1409.6610*.

Vagle, M. D. (2016). *Crafting phenomenological research*: Routledge.

Vaidya, A. (2014). Does dad know best, is it better to do less or just be safe? adapting scaling agile practices into the enterprise. *PNSQC. ORG*, 1-18.

van Muijen, & Jaap, J. (1999). Organizational culture: the focus questionnaire. *European Journal of Work and Organizational Psychology, 8*(4), 551-568.

van Valkenhoef, G., Tervonen, T., de Brock, B., & Postmus, D. (2011). Quantitative release planning in extreme programming. *Information and software technology*.

Van Veenendaal, E. (2008). *Foundations of software testing: ISTQB certification*: Cengage Learning EMEA.

Venkatesh, V., Brown, S. A., & Bala, H. (2013). Bridging the qualitative-quantitative divide: Guidelines for conducting mixed methods research in information systems. *MIS quarterly, 37*(1), 21-54.

Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: four longitudinal field studies. *Management science, 46*(2), 186-204.

Venkatesh, V., Morris, M., Davis, G., & Davis, F. (2003). User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly, 27*(3), 425-478.

Venkatesh, V., Thong, J. Y., & Xu, X. (2012). Consumer acceptance and use of information technology: extending the unified theory of acceptance and use of technology. *MIS quarterly*, *36*(1), 157-178.

VersionOne. (2011). State of Agile Development, from http://www.versionone.com/state_of_agile_development_survey/11/

VersionOne. (2013). State of Agile Survey 2013 Retrieved 21/02, 2015, from http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

VersionOne. (2015). State of Agile Survey 2015 Retrieved 15/11/2016, 2016, from http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf

VersionOne. (2016). 11th Annual State of Agile Report Retrieved 12/04/2017, from https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2

VersonOne. (2011). State of Agile Survey 2011 Retrieved 02/02, 2015, from http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf

Victor, R. (2003). Iterative and incremental development: A brief history. *IEEE Computer Society*.

Vijayasarathy, L., & Turk, D. (2008). Agile software development: A survey of early adopters. *Journal of Information Technology Management*, *19*(2), 1-8.

Vijayasarathy, L., & Turk, D. (2012). Drivers of agile software development use: Dialectic interplay between benefits and hindrances. *Information and Software Technology*, *54*(2), 137-148.

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and software technology*, *53*(1), 58-70.

Wallace, L. G., & Sheetz, S. D. (2014). The adoption of software measures: A technology acceptance model (TAM) perspective. *Information & Management*, *51*(2), 249-259.

Walsham, G. (1993). *Interpreting information systems in organizations*: John Wiley & Sons, Inc.

Wan, J., & Wang, R. (2010). Empirical Research on Critical Success Factors of Agile Software Process Improvement. *JSEA, 3*(12), 1131-1140.

West, D. (2011). Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today. *Forrester, July, 26*.

West, D. (2015). From Controlled Chaos to Differentiation: why companies need to integrate the software lifecycle. *Software Development Times*.

West, D., & Hammond, J. S. (2010). The Forrester Wave™: Agile Development Management Tools, Q2 2010. *Forrester Research*.

Wilkinson, L. (1999). Statistical methods in psychology journals: Guidelines and explanations. *American psychologist, 54*(8), 594.

Williams, B., Onsman, A., & Brown, T. (2010). Exploratory factor analysis: A five-step guide for novices. *Australasian Journal of Paramedicine, 8*(3).

Williams, L., Krebs, W., Layman, L., Antón, A., & Abrahamsson, P. (2004). Toward a framework for evaluating extreme programming. *Empirical Assessment in Software Eng.(EASE)*, 11-20.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM, 14*(4), 221-227.

Wolff, E., & Johann, S. (2015). Technical Debt. *Software, IEEE, 32*(4), 94-c93.

Wood, K., Parsons, D., Gasson, J., & Haden, P. (2013). *It's never too early: pair programming in CS1*. Paper presented at the Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136.

Woods, W. A. (1999). Still No Silver Bullet: Managing and Improving the Software Development Process. [Article]. *Armed Forces Comptroller, 44*(1), 33.

Yang, C., Liang, P., & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software, 111*, 157-184.

Yasar, H., & Kontostathis, K. (2016). Where to Integrate Security Practices on DevOps Platform. *International Journal of Secure Software Engineering (IJSSE), 7*(4), 39-50.

Yin, R. K. (1981). The case study crisis: Some answers. *Administrative science quarterly, 26*(1), 58-65.

Yourdon, E. (1995). When good enough software is best. *Software, IEEE, 12*(3), 79-81.

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and its practices. *IEEE Software, 33*(3), 32-34.

APPENDIX A: SEMI STRUCTURED INTERVIEW GUIDE

INFORMATION SHEET AND CONSENT TO PARTICIPATE IN RESEARCH

KZN HUMANITIES AND SOCIAL SCIENCES RESEARCH ETHICS COMMITTEE (HSSREC)

APPLICATION FOR ETHICS APPROVAL
For research with human participants

Date:

Greetings,

My name is Sanjay Ranjeeth (Student No. 972170992) and I am currently studying for a Doctor of Philosophy (PhD) degree at the University of KwaZulu-Natal (UKZN), in the School of Management, Information Technology and Governance. The discipline of my study is in Information Technology (IT). The contact details for myself as well as my supervisor and the academic department at UKZN are listed below:

Researcher Name: Sanjay Ranjeeth; e-mail: ranjeeths@ukzn.ac.za ;
Office Contact Number: +27 33 260 5641
Mobile Contact Number: +27 84 4768088

Supervisor Name: Professor M Maharaj; e-mail: maharajms@ukzn.ac.za ;
Office contact Number: +27 33 031 – 260 8003
Department of Information Systems & Technology: +27 33 260 5704; + 27 31 260 7051

You are being invited to consider participating in a study that involves research on current practice with regards to the software development process. The objective of the study is to make a contribution by leveraging off the knowledge from current software practitioners in order to propose a framework that guides the implementation of software process models in general. The study does however, have a specific on focus on Agile Software Development Methodology (ASDM).

An Agile Based Integrated Framework for Software Development

The current aspect of the study is directed at obtaining an insight into your experiences of the software development process. This insight will be guided by a semi-structured interview that will be used to add structure to a conversation regarding your experience of software development as well as your perspectives on the current methods and methodologies used for software development. A significant part of your contribution towards this research effort will be in the form of your opinion regarding the use of an agile based approach to develop software. The duration of your participation if you choose to participate and remain in the study is expected to be approximately 40 minutes.

We envisage that the information that you provide will be pivotal in developing a framework that will guide the implementation of ASDM. It is also envisaged that the outcome of the study will make an academic and practitioner-based contribution to the general discourse on ASDM.

This study has been ethically reviewed and approved by the UKZN Humanities and Social Sciences Research Ethics Committee (approval number_____).

In the event of any problems or concerns/questions you may contact the researcher by making use of any of the contact details provided above, or by contacting the UKZN Humanities & Social Sciences Research Ethics Committee. The contact details are as follows:

HUMANITIES & SOCIAL SCIENCES RESEARCH ETHICS ADMINISTRATION
Research Office, Westville Campus
Govan Mbeki Building
Private Bag X 54001
Durban 4000 KwaZulu-Natal, SOUTH AFRICA
Tel: 27 31 2604557- Fax: 27 31 2604609
Email: HSSREC@ukzn.ac.za

Your participation in the study is voluntary and by participating, you are granting the researcher permission to use your responses. You may refuse to participate or withdraw from the study at any time with no negative consequence. There will be no monetary gain from participating in the study. Your anonymity will be maintained by the researcher and the School of Management, I.T. & Governance and your responses will not be used for any purposes outside of this study.

All data, both electronic and hard copy, will be securely stored during the study and archived for 5 years. After this time, all data will be destroyed.

If you have any questions or concerns about participating in the study, please contact me or my research supervisor at the numbers listed above.

Sincerely

SRanjeeth

Sanjay Ranjeeth

CONSENT TO PARTICIPATE

I ………………………………………………………………………………………. (Name) have been informed about the study entitled: *An Agile Based Integrated Framework for Software Development* by *Sanjay Ranjeeth.*

I understand the purpose and procedures of the study.

I have been given an opportunity to ask questions about the study and have had answers to my satisfaction.

I declare that my participation in this study is entirely voluntary and that I may withdraw at any time without affecting any of the benefits that I usually am entitled to.

I have been informed about any available compensation or medical treatment if injury occurs to me as a result of study-related procedures.

If I have any further questions/concerns or queries related to the study I understand that I may contact the researcher at the details provided in **Page 1** of this document.

If I have any questions or concerns about my rights as a study participant, or if I am concerned about an aspect of the study or the researchers then I may contact:

HUMANITIES & SOCIAL SCIENCES RESEARCH ETHICS ADMINISTRATION
Research Office, Westville Campus

Govan Mbeki Building

Private Bag X 54001
Durban
4000
KwaZulu-Natal, SOUTH AFRICA
Tel: 27 31 2604557 - Fax: 27 31 2604609
Email: HSSREC@ukzn.ac.za

I hereby provide consent to:

Audio-record my interview / focus group discussion YES / NO
Video-record my interview / focus group discussion YES / NO
Use of my photographs for research purposes YES / NO

____________________ ____________________
Signature of Participant **Date**

General Instructions for the Interview

During the interview, you are at liberty to request clarification or repetition of the question. There is no time limit set for answering a particular question or for the duration of the interview session. It is advisable to complete the interview in a single sitting.

Demographic & Background Information:

Job Title/Position						
Type of Organisation						
Job Description						
Department						
Gender	*Male*			*Female*		
Qualification(s)	*Under_ graduate Degree/ Diploma*	*Post_ graduate Degree*	*Honours*	*Masters*	*PhD*	*Others- (please specify)*
Approximately how long have you been involved in software development?						

Approximately how long have you been involved in the use of Agile Software Development?	

Pre-Questionnaire for the Interview (Attitude towards Software Development Methods):

1. **The** Waterfall software process model (SPM**), which entails a linear progression from requirements to analysis, design, development & testing, is a viable strategy for the development of software systems.**

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

2. **The** iterative and incremental approach**, which is a non-linear strategy that entails iteration through the phases of the software development lifecycle to produce software incrementally, is a viable strategy for the development of software systems.**

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

3. **What is your opinion on the importance of using the following** analysis and design models**?**

	Very Important	Important	Somewhat Important	Not Important	I Do not use it
Data Flow Diagrams					
Entity Relationship Modelling					
Structure Chart					
User Stories					
Use Case Modelling					

Class Diagrams					
Sequence Diagrams					

4. **What is your opinion on the importance of using a** work/workflow visualization tool **such as the Kanban Board (uses the: *to do, doing and done* columns)?**

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

5. **The** Big Design Up-front (BDUF) **approach to systems modelling enables the development of quality software systems.**

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

6. **How important do you think that using** eXtreme Programming (XP) **methods are?**

	Very Important	Important	Somewhat Important	Not Important	Not Sure/ Not applicable
Pair Programming (PP)					
Test Driven Development (TDD)					
Availability of an on-site customer					
Continuous Integration					
Code Re-factoring					

7. **How important are the following** Scrum **based methods in the software development process?**

	Very Important	**Important**	**Somewhat Important**	**Not Important**	**Not Sure/ Not applicable**
Product Backlog (PB)					
A Sprint					
Daily Scrum meetings					
Time-Boxing					
Sprint Backlog					
Sprint Review					
Sprint Retrospective Meeting					
Burn Down Chart					

Part B of the Questionnaire

The Main Interview Questions:

In this part of the interview, the questions are directed at your **general perception of the processes/methods used for ASDM** based on your experiential knowledge of ASDM (or of software development in general).

PART A – ***PHENOMENOLOGICAL KNOWLEDGE OF AGILE SOFTWARE DEVELOPMENT Methodology (ASDM) – A Bracketing Approach***	
1	Main Please provide some detail regarding your **experience(s) of the** processes/methods used with ASDM (or software development in general); describe any experience that you have had with ASDM.

2.	Main What is your view of? • **ASDM – *project type/scalability* *(Probe)*** • **Software development in general**

Context for **PART B** of the Interview (Knowledge & Practice):

In this part of the interview, the questions are directed at your perceptions of the **generic activities involved in the SOFTWARE (DEVELOPMENT) PROCESS**.

<table>
<tr><th colspan="2">PART B – <em>PHENOMENOLOGICAL KNOWLEDGE OF GENERAL SOFTWARE DEVELOPMENT METHODOLOGY(SDM) – A Hermeneutic Approach</em></th></tr>
<tr><td><strong>1.</strong></td><td>Main What do you <strong>think</strong> of the <strong>iterative and incremental (IID)</strong> approach to software development as opposed to the Waterfall approach?
<em>A follow-up question (optional):</em> IID endorses the delivery of the system with incrementally greater functionality in each Iteration. What is your opinion/experience of delivering system functionality in a “<strong>piecemeal</strong>” kind of manner?</td></tr>
<tr><td><strong>2.</strong></td><td><em>The use of a Big Design Up Front (<strong>BDUF</strong>) strategy has been claimed to be problematic because it slows down the pace of development and impedes the prospect of refining/changing the system requirements.</em> Main Comment on this criticism of the BDUF approach…follow up question: Based on your experience of analysis and design modelling, which analysis and design models do you feel are pivotal to the software process? <em>Probe:</em> Sprint 0;</td></tr>
<tr><td><strong>3.</strong></td><td>Main In your experience of software development, how effective are the <strong>object oriented approach (OO)</strong>, the <strong>classical/structured approach</strong> and the <strong>hybrid approach (combination of OO and classical)</strong> to systems modelling?
<em>A follow-up question (optional): Probe:</em> What would you regard as an <strong>optimal mix</strong> of these software development approaches?</td></tr>
</table>

Context for **PART C** of the Interview:

In this part of the interview, the questions are directed at your perceptions of the activities intrinsic to the implementation of ASDM with a specific focus on **eXtreme Programming (XP) and Scrum** methodologies.

<table>
<tr><th colspan="2">PART C – <em>PHENOMENOLOGICAL KNOWLEDGE OF AGILE SOFTWARE DEVELOPMENT METHODOLOGY(ASDM) – A Hermeneutic Approach</em></th></tr>
<tr><td><strong>1.</strong></td><td><em>ASDM advocates a preference for <strong>quick/early release of working software</strong> as opposed to ensuring that the software system has the pre-requisite documentation in place before it can be released to the user community.</em> Main Based on your experience of software development in general, what is your comment on the following?
• <strong>Software release with</strong> incomplete documentation</td></tr>
</table>

	• **Software release with** *incomplete testing/ availability of on-site customer/ operations staff/Continuous Integration/Deployment/ Quest for Business Value* (*Probe*)
2.	Main *From your experience of ASDM or general software development*, how would you recommend that the changes to the **requirements specification be accommodated** once the software process commences? *– scope issues* Follow up: At what point in the development cycle would you recommend that there should be no further changes to the requirements specification for the system? *Probe: Opinion on: Agile & Project Management; DevOps, UX Design, Usefulness of Jira Scrum/KanBan Story Board*
3.	*Academic sources have suggested that:* • ***the Waterfall Methodology is too** prescriptive and documentation-centric,* • ***XP enhances the prospect of developing** quality systems* • ***Scrum enables** better management of the software process.* Main Which of these qualities would you prioritise? Why? Follow up: Do you think that **XP methods could be integrated with Scrum methods** and possibly with aspects of the **Waterfall methodology (as well as KanBan, Lean, FDD…)**? Why did you respond in this way? *Probe:*

Context for **PART D** of the Interview:

In this part of the interview, the questions are directed at your suggestions for the implementation of ASDM within the context of specific strands of **organisational culture (OC)**. This part of the questionnaire is aligned to the imperative to provide guidance on the implementation of ASDM so that it resonates quite well with the culture that prevails in an organisation.

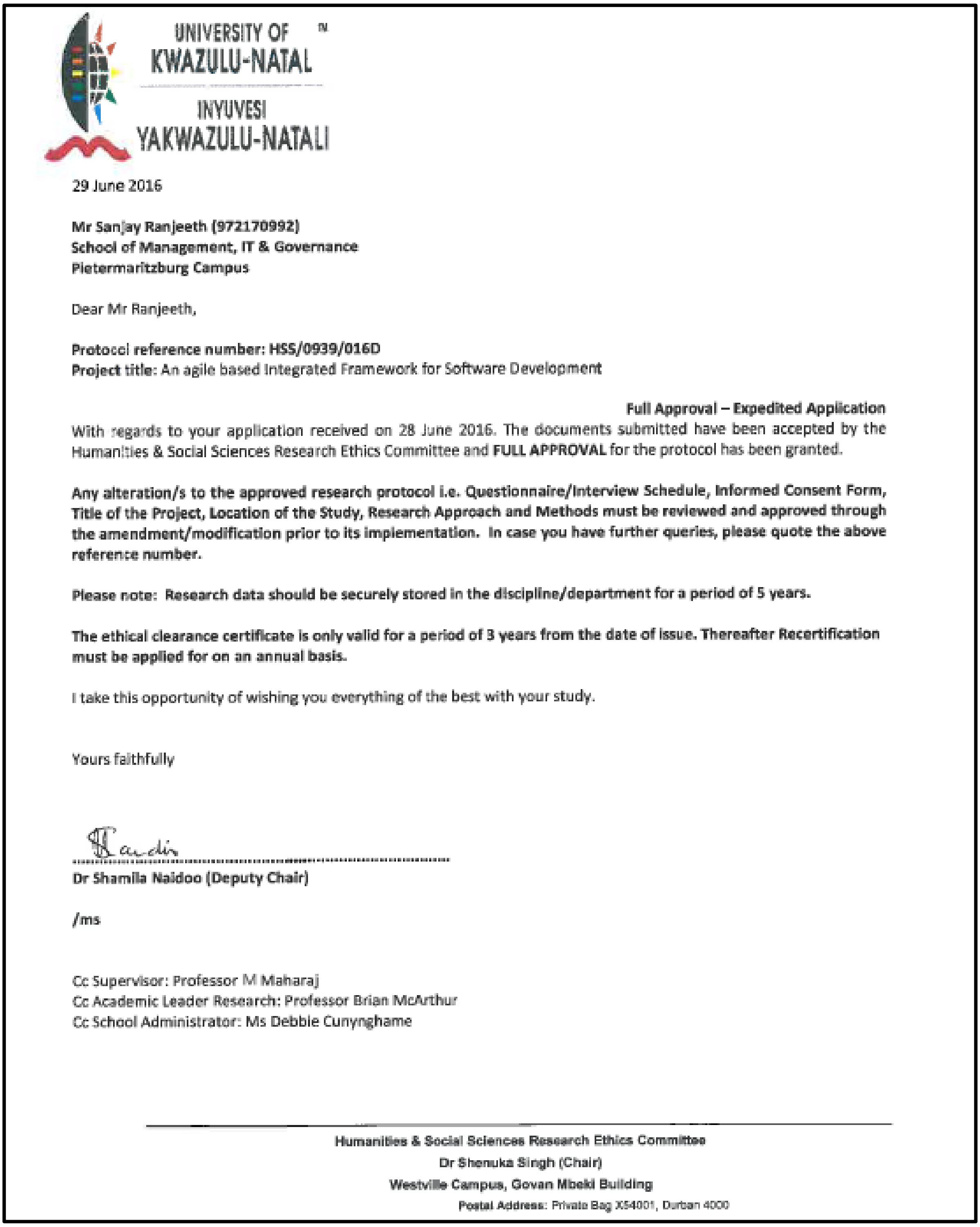
PART E – *PHENOMENOLOGICAL KNOWLEDGE OF THE INFLUENCE OF ORGANISATIONAL CULTURE ON THE SOFTWARE PROCESS/ASDM IN THE CONTEXT OF THE CVF.*

A Hermeneutic Approach

1.	*There have been claims within the academic and practitioner community that ASDM has the potential to de-generate into a code and fix/hacker mentality where the developers are (possibly) entrusted with too much autonomy. In such instances, ASDM may not be successful because of the organisational culture. The academic community have also suggested that the culture within an organisation may be classified according to a theoretical model named the CVF that distinguishes between* ***4 types of OC****.* ***In this portion of the interview, I will provide you with a few significant characteristic(s) of each cultural types and you could make a suggestion(s) regarding a software process model/software methods that resonate with the specific strand of OC.*** Main ***What kind of culture would enable Agile Methodology to achieve optimal success?*** Follow up: ***What kind of management support would enable ASDM to thrive?*** • Developmental Culture: **An organisation that is** *quite liberal* ***in its stance towards product development, embraces*** *risk taking, focuses broadly about the big picture and big ideas* ***and are keen to use innovative thinking to establish competitive advantage.*** • Rational Culture**: An organisation where there is a strong focus on** ***achieving*** *high productivity, enabling innovation with economic consumption of resources* ***– basically a “****bang for bucks****”*** **culture.** • Group Culture. **An organisation that has a strong focus on** *maintaining traditions and norms and values* ***that have contributed to the success of the organisation*** **in the past.** • Hierarchical Culture: **An organisation that has a strong focus on** *management control,* ***security, accountability, a rules-based organisation where*** *predictability is valued over innovation*.
2.	Main Do you think that there is a need to align your software development methodology with the culture that prevails in an organisation? Follow up: How can agile methodology be used in a way that enables alignment to the prevailing OC?

Thank You!

APPENDIX B: ETHICAL CLEARANCE PHASE 1



UNIVERSITY OF KWAZULU-NATAL
INYUVESI YAKWAZULU-NATALI

29 June 2016

Mr Sanjay Ranjeeth (972170992)
School of Management, IT & Governance
Pietermaritzburg Campus

Dear Mr Ranjeeth,

Protocol reference number: HSS/0939/016D
Project title: An agile based Integrated Framework for Software Development

Full Approval – Expedited Application

With regards to your application received on 28 June 2016. The documents submitted have been accepted by the Humanities & Social Sciences Research Ethics Committee and **FULL APPROVAL** for the protocol has been granted.

Any alteration/s to the approved research protocol i.e. Questionnaire/Interview Schedule, Informed Consent Form, Title of the Project, Location of the Study, Research Approach and Methods must be reviewed and approved through the amendment/modification prior to its implementation. In case you have further queries, please quote the above reference number.

Please note: Research data should be securely stored in the discipline/department for a period of 5 years.

The ethical clearance certificate is only valid for a period of 3 years from the date of issue. Thereafter Recertification must be applied for on an annual basis.

I take this opportunity of wishing you everything of the best with your study.

Yours faithfully

Dr Shamila Naidoo (Deputy Chair)

/ms

Cc Supervisor: Professor M Maharaj
Cc Academic Leader Research: Professor Brian McArthur
Cc School Administrator: Ms Debbie Cunynghame

Humanities & Social Sciences Research Ethics Committee
Dr Shenuka Singh (Chair)
Westville Campus, Govan Mbeki Building
Postal Address: Private Bag X54001, Durban 4000

APPENDIX C: INTERVIEW SCHEDULE

	Interview Identifier & Name of Interviewee	Organisation type(s) – past & present	Capacity (Past & present)	Years of Experience		Interview Type & Duration
				General Software Develop_ ment	Agile Methodology	
1	1 [redacted]	Investment /Insurance	Developer/ Analyst manager/BA	15	8	Face to face (52 minutes)
2	2 [redacted]	Bespoke Software Solutions	Developer Manager	10	7	Skype Video (45 minutes)
3	3 [redacted]	Banking	Head of Online Banking/ BA	8	5	Skype Video (45 minutes)
4	4 [redacted]	Software Solutions Provider	Software Engineer/ Consultant	8	6	Face to Face (60 minutes)
5	5 [redacted]	Banking	Developer Systems Analyst	5	3	Skype Video (45 minutes)
6	6 [redacted]	Banking	Developer/ Chief IT security officer	5	2	Skype Video (45 minutes)
7	7 [redacted]	Banking	Developer/ Systems Analyst	9	7	Face to Face (60 minutes)
8	8 [redacted]	National Logistics Provider	Developer Systems Analyst manager/ Solutions Architect	7	6	Skype Video (55 minutes)
9	9 [redacted]	Banking	Software Developer/ Team Leader	5	2	Skype Video (35 minutes)
10	10 [redacted]	Agriculture	Developer Systems Analyst manager	30	9	Face to Face (55 minutes)
11	11 [redacted]	Banking	Chief Software Methodologist	12	8	Face to Face (60 minutes)
12	12 [redacted]	Banking/ Motor Vehicle	BA/Systems Architect	7	5	Face to Face (60 minutes)

13	13 [redacted]	Petro-chemical	Manager/ Developer	7	5	Face to Face (60 minutes)
14	14 [redacted]	Banking	Build Engineer/ DevOps Team Leader	14	5	Skype Video (50 minutes)
15	15 [redacted]	Banking	Software Engineer	10	8	Face to Face (60 minutes)
16	16 [redacted]	Software Solutions	Developer/ Manager/Solution Architect	10	7	Face to Face (60 minutes)
4 Interviews NOT included in the Exploratory Phase of Qualitative Data Analysis						
17	17 Grady Booch	Software Engineering Consultant	Software Engineer and IBM Research Fellow	40	15	Skype Video (60 minutes)
18	22 Bob Aiello	Software Consultancy	Build Engineer/ IEEE Chairperson of Working Group AgileSA	20	12	Skype Video (45 minutes)
19	23 Brad Black	Software Consultancy	Agile Coach	15	12	Skype Video (65 minutes)
20	24 Jonathan Frankel	Banking	DevOps Team Leader	12	8	Skype Video (60 minutes)

APPENDIX D: SURVEY QUESTIONNAIRE

UKZN HUMANITIES AND SOCIAL SCIENCES RESEARCH ETHICS COMMITTEE (HSSREC)

Information Sheet and Consent to Participate in Research

Date: 13th October 2017

Greetings- My name is Sanjay Ranjeeth (Student No. 972170992) and I am currently studying for a Doctor of Philosophy (PhD) degree at the University of KwaZulu-Natal (UKZN), in the School of Management, Information Technology and Governance. The contact details for myself as well as my supervisor and the academic department at UKZN are listed below:

Researcher Name: Sanjay Ranjeeth;
e-mail: ranjeeths@ukzn.ac.za ;
Office Contact Number: +27 33 260 5641
Mobile Contact Number: +27 84 4768088

Supervisor Name: Professor M Maharaj;
e-mail: maharajms@ukzn.ac.za ;
Office contact Number: +27 33 031 - 260 8003
Department of Information Systems & Technology:
+27 33 260 5704; + 27 31 260 7051

You are being invited to consider participating in a **follow-up study** that entails research on the acceptance of an agile based framework for software development. The framework is named the Scrum Development Operations Model (SDOM) and is part of a study titled:

An Agile Based Integrated Framework for Software Development

SDOM represents a convergence of one aspect of the empirical data that was gathered as part of the first phase of the study. The "first phase" empirical data consisted of interviews with experienced software practitioners in South Africa who have provided their insight into the issues related to the methodology used for software development in South African organisations.

The current phase of the study consists of a survey that is aligned to a theoretical framework that guides knowledge on the acceptance by software practitioners of a software development methodology. **The duration of your participation if you choose to participate in this phase of the study is expected to be approximately *15 minutes* for the filling-in of the survey questions.**

This study has been ethically reviewed and approved by the UKZN Humanities and Social Sciences Research Ethics Committee (approval number: HSS/0939/016D).

In the event of any problems or concerns/questions you may contact the researcher by making use of any of the contact details provided above, or by contacting the UKZN Humanities & Social Sciences Research Ethics Committee. The contact details are as follows:

HUMANITIES & SOCIAL SCIENCES RESEARCH ETHICS ADMINISTRATION

Research Office, Westville Campus, Govan Mbeki Building, Private Bag X 54001
Durban 4000 KwaZulu-Natal, SOUTH AFRICA, Tel: 27 31 2604557- Fax: 27 31 2604609

Email: HSSREC@ukzn.ac.za
Your **participation in the study is voluntary** and by participating, you are granting the researcher permission to use your responses. **Your anonymity will be maintained** by the researcher and the School of Management, I.T. & Governance and your responses will not be used for any purposes outside of this study.
All data, both electronic and hard copy, will be securely stored during the study and archived for 5 years. After this time, all data will be destroyed.
If you have any questions or concerns about participating in the study, please contact me or my research supervisor at the numbers listed above.

Sincerely

SRanjeeth

Sanjay Ranjeeth

CONSENT TO PARTICIPATE

I .... (Name),
have been informed about the study entitled: *An Agile Based Integrated Framework for Software Development* by *Sanjay Ranjeeth.*
I understand the purpose and procedures of the study.

I have been given an opportunity to ask questions about the study and have had answers to my satisfaction. I declare that my participation in this study is entirely voluntary and that I may withdraw at any time without affecting any of the benefits that I usually am entitled to.
If I have any further questions/concerns or queries related to the study I understand that I may contact the researcher at the details provided in **Page 1** of this document.

__________________________ ____________________
Signature of Participant **Date**

Demographic & Background Information:

Job Title/Position		
Type of Organisation		
Job Description		
Department		
Gender	*Male*	*Female*
Approximately how long have you been involved in software development?		
Approximately how long have you been involved in the use of Agile Software Development Methodology?		

Introduction

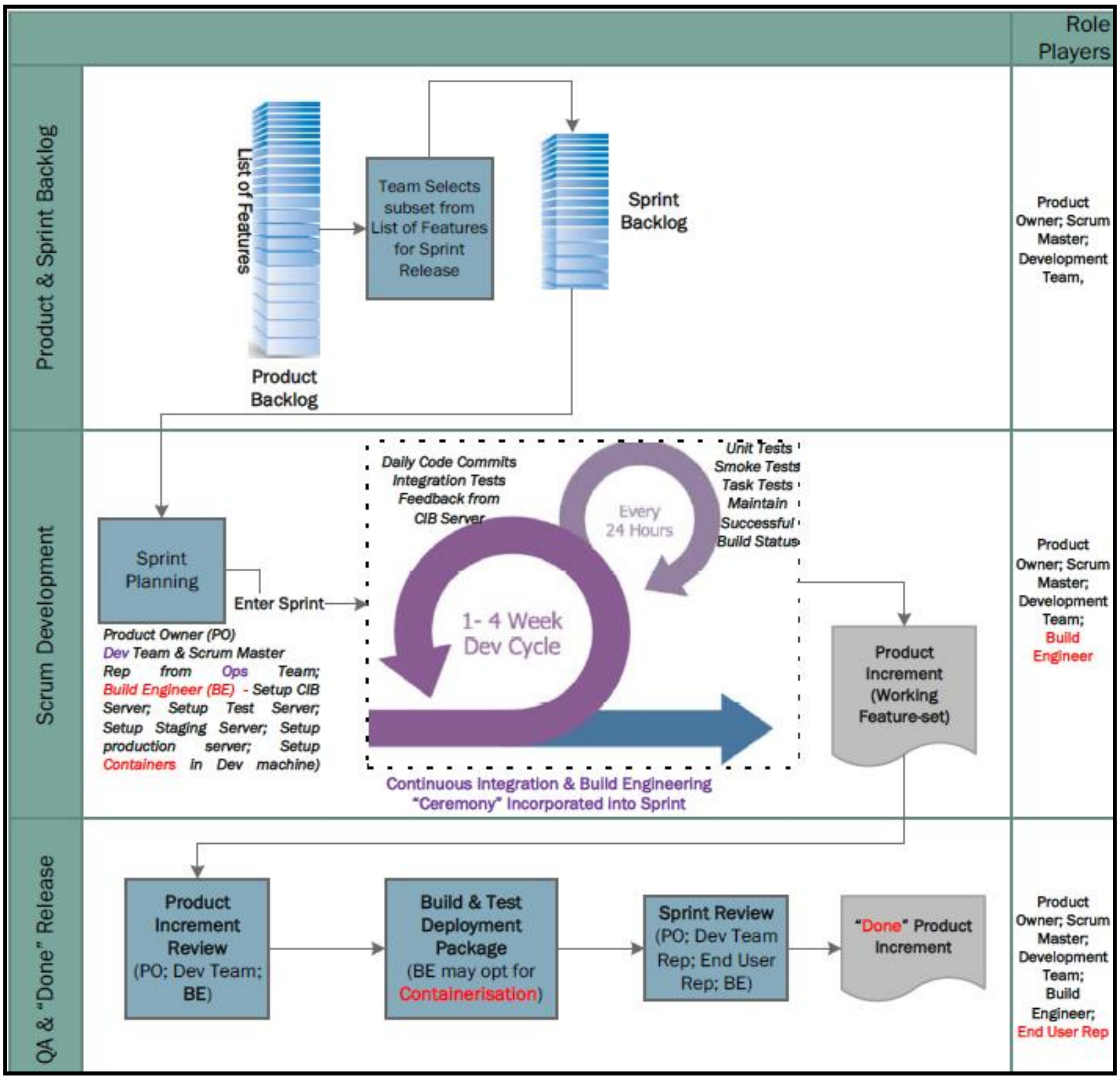
This questionnaire has been developed in order to obtain feedback on the degree of acceptance of the proposed re-engineering of the Agile based Scrum methodology for software development. This is a follow-up to the first part of a study that has gathered empirical evidence that attests to the following outcomes:

- Scrum has been endorsed as the de-facto methodology for software development in South African business organisations
- Software practitioners in these organisations are quite comfortable with the "inner workings" of Scrum based software development methodology
- There is a concern that the Agile imperative of *delivering working software that yields a quick return of* ***business value*** is not being upheld
- A significant reason for the afore-mentioned phenomenon is that there is a "bottle-neck" created because of a lack of focus with regards to the operations/infrastructure requirements of the software systems that are developed using Scrum methodology

In an effort to resolve this situation the current study has proposed an integrated model for software development that is centred on Scrum processes and is integrated with roles and activities for operations/infrastructure staff members.

The proposed model is named the **Scrum Development Operations Model (SDOM)**.

Please do a review of SDOM and provide a response via the structured questionnaire indicating your acceptance of SDOM. An illustration and detailed narrative of the SDOM may be accessed at: http://143.128.146.30/SDOM/ScrumOps/SDOMIntro.aspx where you are invited to place a few comments. An illustration of SDOM is provided for your quick reference.


Role Players
Product & Sprint Backlog
List of Features
Team Selects subset from List of Features for Sprint Release
Sprint Backlog
Product Backlog
Product Owner; Scrum Master; Development Team,
Scrum Development
Sprint Planning
Enter Sprint
Product Owner (PO) Dev Team & Scrum Master Rep from Ops Team; Build Engineer (BE) - Setup CIB Server; Setup Test Server; Setup Staging Server; Setup production server; Setup Containers in Dev machine)
Daily Code Commits Integration Tests Feedback from CIB Server
Every 24 Hours
Unit Tests Smoke Tests Task Tests Maintain Successful Build Status
1- 4 Week Dev Cycle
Product Increment (Working Feature-set)
Continuous Integration & Build Engineering "Ceremony" Incorporated into Sprint
Product Owner; Scrum Master; Development Team; Build Engineer
QA & "Done" Release
Product Increment Review (PO; Dev Team; BE)
Build & Test Deployment Package (BE may opt for Containerisation)
Sprint Review (PO; Dev Team Rep; End User Rep; BE)
"Done" Product Increment
Product Owner; Scrum Master; Development Team; Build Engineer; End User Rep

Please use an **X** to indicate the most appropriate response

1. Perceived Usefulness (PU) of the Proposed SDOM

PU of SDOM	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Using SDOM will improve the performance of software development teams.					
Using SDOM will increase the productivity of software development teams.					
Using SDOM will improve the quality of the software developed in my organisation					
Using SDOM will make it easier to develop software.					
The advantages of using SDOM outweigh the disadvantages of using SDOM					
SDOM will be useful as a general approach for software development.					

2. Compatibility of the Proposed SDOM

Compatibility of SDOM	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
SDOM is compatible with the way software is developed in my organisation					
SDOM is compatible with the work related responsibilities of software development					
SDOM will 'fit in' well with current software development practice in my organisation.					

3. Subjective Norm/Social Factors that Influence the use of the Proposed SDOM

Subjective Norm/Social Factors	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Influential people in my organisation will endorse the use of SDOM					

People who are important to me will have a positive view of SDOM					
My colleagues will have a positive attitude towards using SDOM.					

4. Perceived Organisational Support for the use of SDOM

Organisational Support	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
My organisation will provide help and resources for the use of SDOM					
SDOM will receive management support in my organisation					

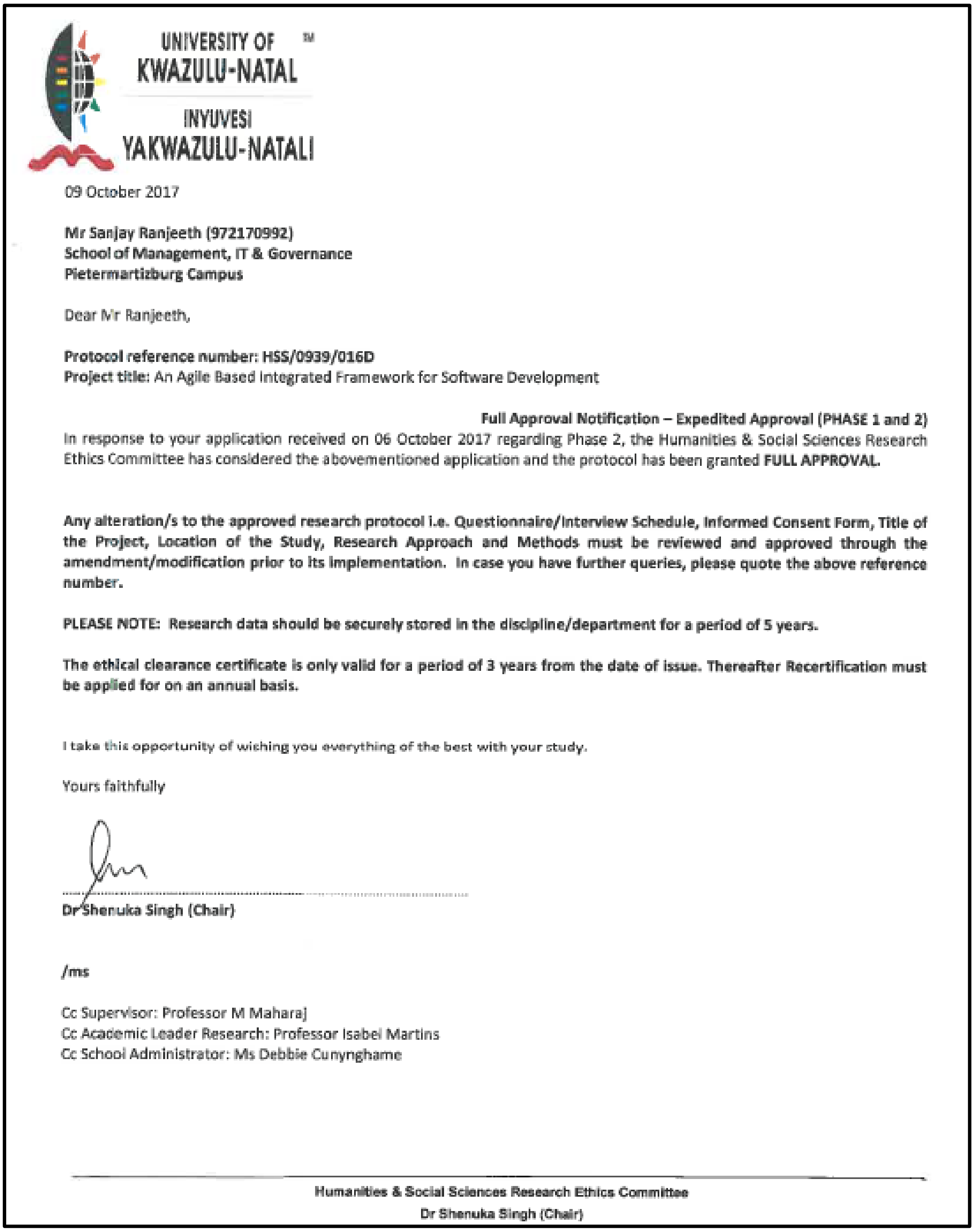
5. Behavioural Intention to use of SDOM

Behavioural Intention	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I think that the SDOM is a model that I will like to try out in my organization.					
Given the opportunity, I would use SDOM					

6. Comments/Feedback/Suggestions on the SDOM

Thank You

APPENDIX E: ETHICAL CLEARANCE PHASE 2



UNIVERSITY OF KWAZULU-NATAL™
INYUVESI YAKWAZULU-NATALI

09 October 2017

Mr Sanjay Ranjeeth (972170992)
School of Management, IT & Governance
Pietermaritzburg Campus

Dear Mr Ranjeeth,

Protocol reference number: HSS/0939/016D
Project title: An Agile Based Integrated Framework for Software Development

Full Approval Notification – Expedited Approval (PHASE 1 and 2)

In response to your application received on 06 October 2017 regarding Phase 2, the Humanities & Social Sciences Research Ethics Committee has considered the abovementioned application and the protocol has been granted **FULL APPROVAL.**

Any alteration/s to the approved research protocol i.e. Questionnaire/Interview Schedule, Informed Consent Form, Title of the Project, Location of the Study, Research Approach and Methods must be reviewed and approved through the amendment/modification prior to its implementation. In case you have further queries, please quote the above reference number.

PLEASE NOTE: Research data should be securely stored in the discipline/department for a period of 5 years.

The ethical clearance certificate is only valid for a period of 3 years from the date of issue. Thereafter Recertification must be applied for on an annual basis.

I take this opportunity of wishing you everything of the best with your study.

Yours faithfully

Dr Shenuka Singh (Chair)

/ms

Cc Supervisor: Professor M Maharaj
Cc Academic Leader Research: Professor Isabel Martins
Cc School Administrator: Ms Debbie Cunynghame

Humanities & Social Sciences Research Ethics Committee
Dr Shenuka Singh (Chair)

APPENDIX F: EMAIL CORRESPONDENCE

On 17 Oct 2017, at 14:45, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:

Hi Lester

I trust you are well!

I must firstly thank your for the invaluable insight that I obtained from my conversation with you.
The reference that you provided for the bi-modal approach to software development has received wide coverage in my PhD write-up.
I am currently trying to obtain a survey based response to one of the models that have been developed in my study. It will be really appreciated in you could fill in the attached **15 minute** survey on the Word document and return to me. Also, you can simply type in your name/ jpeg the signature - there is no need to scan the document and if there is any further time that you have at your disposal, please leave a comment at the study's website at: http://143.128.146.30/SDOM/ScrumOps/SDOMIntro.aspx

Once more thanks for your support and highly valued input - enjoy the rest of your day:)

On 29 Oct 2017, at 12:22, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:

Hi Lester ...thanks for your invaluable insight. I'm really appreciative of the critical insight that you provide.
If you could "*humour*" me just for a short while with regards to the main points that you have raised, it will be highly appreciated...so here goes just a short response:)

Your comments regarding SAFe are well intended and accepted. I have actually done a critical review of the SAFe approach and if it can be achieved then that will be great...however, the transition to SAFe requires a complete shift in the culture of the organisation which I have established is not going to be easy to achieve. My proposal is to start off in the "small" in order to rectify the immediate problems that are an impediment to continuous delivery and deployment and from the empirical evidence that I have gathered, it all points to a lack of co-operation/co-ordination between the development teams and the build engineer or systems architect. The development and deployment environments are not identical resulting in a huge "bottleneck" situation when it comes to deployment. The SAFe approach may rectify this problem … owever the SDOM model that I propose with the strategy of containerisation and the active involvement of the build engineer with the agile teams will at least contribute to an alleviation of the "bottleneck" situation...the idea behind proposing such models is that they are then subjected to tests of validity in future studies ...so maybe a start has to be made somewhere so that the agile initiative is enhanced on the basis of experiential knowledge.

DevOps approaches as practitioners are calling it. This is very hard to do and I've only seen early startups being able to adopt this approach because they have no baggage. Mature organisations have an *extremely* hard time putting in DevOps practises due to substantial investment and hardwired SDLC over many years (Conways Law prevails here as teams and processes are structured around Org structures and politics in the organisation) including the mentality and culture resistance to this new way fo working.(which is the hardest obstacle to overcome, as you also highlight)

I hope that this response alleviates your concerns somewhat...although I respect your knowledge and authority over this subject domain. Please feel free to provide me with further comments and feedback at your convenience because it is this kind of critical insight that I really appreciate!

Once more thanks for your time and contribution to this discourse on the improvement of software engineering processes in South Africa...have a great day:)

Regards
Sanjay Ranjeeth
ranjeeths@ukzn.ac.za

From: Lester Masher <lmasher@icloud.com>
Sent: Sunday, 29 October 2017 1:32 PM
To: Sanjay Ranjeeth
Subject: Re: A Quick Favour

Right on track…quantitative approaches to measure software development effectiveness has eluded us.

Have you looked at **MBSE** (Model-Based System Engineering approaches) which are now seeping their way into mainstream development as **"Low-Code"** development. Model driven design has been around for ages since early CASE tools and then picked up by **OMG** with the conception of UML etc. and then late in 2000's we saw hope again with the advent of "**Domain Driven"** design by fowler and others. But the utopia still eludes us.
However, now after years of hiatus and with newer with AI techniques (maturing exponentially) it seems the technology is coming if age and we are getting closer to realising the dream of M*odel driven development.* Eventually though automate/augmented coding by AI systems and readily available frameworks and patterns that can be applied to any software problem by AI systems is the ultimate goal, and with the speed of Ai those horizons are getting shorter and shorter….

Regards

On 29 Oct 2017, at 14:00, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:
Right now ...it does sound kind'a utopic ...like you mentioned...however I can see the value in embracing this approach conceptually because the impediments are quickly being eroded by the great strides made in AI...however, the assumptions underlying this approach (such as standardisation of modelling nomenclature) needs to be tested ...ironically the MBSE approach is a contradiction of agility because of heavy reliance on upfront modelling but at the same time it is aligned to the agile principle that enables the delivery of working software quite quickly...provided you have a mature/sophisticated model base to work from...I suppose it may work well with the service oriented architecture (SOA) approach so that organisations have a ready-made set of underlying services that are aligned to the business models thereby enabling really low code development.

On 29 Oct 2017, at 14:20, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:

Thanks for those references Lester - I've actually attached quite a few references to the contributions made by Jez Humble in my dissertation...and I am also trying to set up an interview with him.

However, I was able to speak to a colleague of his by the name of Bob Aiello who is the chairperson of the IEEE working group on agile improvement techniques and one of the main things that I have observed is that much of the knowledge acquired is knowledge is based on anecdotal evidence. I did however subject this to some form of empirical testing as soon as I realised that there is a problem between development and deployment. So the model that I propose is based on the suggestions coming from some of the experienced software engineers who were asked about an integration of operations expertise into the development domain ...and this is pretty much what I illustrated in the SDOM that I proposed ...it's not "rocket science" but it is envisaged that such an integrative/collaborative approach will eventually evolve into a full blown SAFe environment...it is also based on empirical data that I have collected...which is not easy to acquire in the software development environment because of the rapid rate of change ...so one of the biggest challenges that I faced was that of the "moving target problem" ...solving problems of a technological nature requires a quick and dynamic approach...not part of a PhD dissertation...but these are lessons that you learn as you become acquainted with the territory:(

Regards
Sanjay Ranjeeth

From: Lester Masher <lmasher@icloud.com>
Sent: 29 October 2017 02:30 PM
To: Sanjay Ranjeeth
Subject: Re: A Quick Favour

I was just thinking aloud now Sanjay, but if you can determine the correct leveraged data points to measure software development effectiveness in the SDOM model and you can also provide large and constant data sets (by analysing enough software projects) then you can build a deep learning model to automatically find the areas for optimisation. In that way you can come up with an optimised SDOM…

Could me in if you want to work on something like that.

On 29 Oct 2017, at 14:17, Lester Masher <lmasher@icloud.com> wrote:

And still further….. if you can collect enough data on actual running process in a business's value chain and operations then an AI can learn how the business operates in real-time and then propose new models. It can then go further by automatically updating the processes for example modifying (BPMN) processes on the fly.
We already starting with early stages to do that by replacing manual processes with "**Robotics**" automation" albeit still with human intervention to build the models in traditional business process management tools. Next step is we point AI tools to running processes in an organisation and it applies deep learning to build better models via process intelligence with a feedback loops to update current running software. This is one way to

still use Legacy systems but learn and modernize them build new optimised software. So don’t throw away the legacy applications just yet….

From: Sanjay Ranjeeth
Sent: Sunday, 29 October 2017 2:33 PM
To: Lester Masher
Subject: Re: A Quick Favour

Great idea Lester ...will definitely include you on any such intervention...and thanks for contributing your time and wisdom...it's most appreciated!

From: Rishi <ashrisba@gmail.com>
Sent: Monday, 06 November 2017 2:06 PM
To: Sanjay Ranjeeth
Subject: Re: Survey Response

Hi Sanjay
Sounds good. I have extended the questionnaire to a few colleagues. I was cautious to extend to those that have some Agile knowledge else you going to get a skewed sample set.
I might send to a digital architecture team within Microsoft - if I get Legal permission then you might make the numbers you require easily.
I told candidates to respond before the end of the week.

I will gladly include you in the comms for the UKZN talk. I am going to write to Ashraf later today - happy to include you in 'cc ?

Always glad to help and yes, will be great to meet in person. Maybe we can meet with Suvash soon? Im seeing him tomorrow morning.

Wishing you well,
-Rishi

On 6 November 2017 at 11:23, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:
Wow...that is brilliant Rishi!

Firstly, please keep me informed of the details regarding your talk at UKZN and if you guys need any assistance with the logistics side of things ...it will also enable me to ensure that there is maximum attendance from the IT students because such an exposure will be invaluable to these students.

Also, it would be great to have the pleasure of meeting you in person and I must offer my thanks and appreciation to you for your generous offer of further assistance...even a single respondent will be great ...if you do have access to more people then perhaps 3 or possibly 4 more people to respond to the survey will be quite a bonus for me right now...I've put a call out to various people in the IT sector but understandably, people just don't have the time anymore ...such is the nature of the hectic professional world of IT as I'm sure you are fully aware of...hence my utmost appreciation to you for your effort to assist:)

On 6 November 2017 at 09:26, Sanjay Ranjeeth <RanjeethS@ukzn.ac.za> wrote:
Hi Rishi

Thanks for taking the time and making the effort to complete the survey. I would just like to endorse many of the comments that you have made and I certainly appreciate your viewpoints knowing full well that these comments come from a highly experienced and qualified source.
Just a quick reply to the issue of agile adoption - you're completely justified with your views on this matter and these viewpoints actually align to a different part of my study. Based on the assumption that agile has been adopted, one of the main areas of concern that many of the developers and business managers expressed was the lack of integration with operations/infrastructure requirements which were not factored into the development life-cycle...and so this aspect of my study may be seen as the start of a "minor" movement in that direction...handling this from an organisational perspective using approaches such as DevOps and SAFe are turning out to be quite a challenge because of prevailing cultures in an organisation...so basically this is an attempt to propose a workable solution that does not have organisational-wide impact until such time an organisation reaches an acceptable level of agile maturity.

Sorry about what turned out to be a long-winded reply ...but I think the time and effort that you invested in applying your expertise to respond to this survey is deserving of a respectful response:)

Once more, I must thank you for affording me the opportunity to obtain an insight into your expert knowledge in this domain ....this is much appreciated and it is an absolute pleasure to make your acquaintance...and thanks for your well wishes ...all the best to you as well:)

Regards
Sanjay Ranjeeth

INDEX