Program management practices in context of Scrum: A case study of two South African software development SMMEs

By

Alveen Singh

Submitted in fulfilment of the requirements for the degree of

Doctor of Technology

in the

Department of Information Technology in the
Faculty of Accounting and Informatics

at the

Durban University of Technology
Durban, South Africa

December 2014

Acknowledgements

First, I would like to express my deepest appreciation to my mentor, Professor Kosheek Sewchurran, who throughout this incredible journey continually and consistently conveyed a spirit of adventure. Without his guidance and his wisdom in the research area this dissertation would not be possible.

Second, I would like to thank my co-mentor, Professor Thiruthlall Nepal, for his initial encouragement and for the *shove* to get me moving with this research.

Thank you to the two software SMMEs that volunteered participation. Without their engagement and commitment this study would not be possible.

A special thank you to my wife, Neerisha Singh, whose unwavering and undiminishing support gave me strength to see this study through to its completion.

Thank you to my boss, Ms Kesarie Singh, who supported me throughout this journey and to my colleagues who encouraged and cheered me on.

Thank you to Mum and Dad for believing in me and for seeing potential that sometimes evaded me.

Supervisor: Professor Kosheek Sewchurran

Co-supervisor: Professor Thiruthlall Nepal

Dedication

To Neerisha who has given me unconditional friendship and love.

To our twins, Aryan and Aarav, whose growth provides a constant source of joy and pride.

Table of Contents

List of Figures

List of Tables

ABSTRACT

Agile approaches have proliferated within the software development arena over the past decade. Derived mainly from Lean manufacturing principles, agile planning and control mechanisms appear minimal and fluid when compared to more traditional software engineering approaches. Scrum ranks among the more popular permutations of agile. Contemporary literature represents a rich source of contributions for agile in areas such as practice guidelines, experience reports, and methodology tailoring; but the vast majority of these publications focus on the individual project level only, leaving much uncertainty and persistent questions in the multi-project space. Questions have recently been raised, by both academics and practitioners alike, concerning the ability of Scrum to scale from the individual project level to the multi-project space.

Program management is an area encompassing practice and research areas concerned mainly with harmonizing the existence of competing simultaneous projects. Existing literature on program management essentially perceives projects as endeavours that can be carefully planned at the outset, and controlled in accordance with strong emphasis placed on economic and schedule considerations. This complexion seems to be mostly a result of well-established and ingrained management frameworks like Project Management Institute (PMI), and is largely at odds with emerging practices like Scrum. This disparity represents a gap in the literature and supports the need for deeper exploration. The conduit for this exploration was found in two South African software development small to medium sized enterprises (SMMEs) practicing Scrum. The practical realities and constraints faced by these SMMEs elicited the need for more dynamic program management practices in support of their quest to maximize usage of limited resources. This thesis examines these practices with the aim of providing new insights into the program management discourse in the context of Scrum software development environments.

The research approach is qualitative and interpretive in nature. The in-depth exploratory case study research employed the two software SMMEs as units of analysis. Traditional ethnographic techniques were commissioned alongside minimal researcher participation in project activities. Activity Theory honed the data analysis effort and helped to unearth the interrelationships between SMME characteristics, program management practices, and Scrum software development. The results of the data analysis are further refined and fashioned into eleven knowledge areas that represent containers of program management practices. This is the product of thematic analysis of literature and data generated from fieldwork. Seeing as the observed practices were highly dynamic in nature, concept analysis provided a mechanism by which to depict them as *snapshots* in time. As a theoretical contribution, proposed frameworks were crafted to show how program management practices might be understood in the context of organizations striving towards agile implementation. Furthermore, representations of the mutually influential interfaces of SMME characteristics and Scrum techniques that initiate the observed fluid nature of program management practices, are brought to the fore.

Keywords: agile, scrum, program management, activity theory, software development, IT project management, SMMEs

CHAPTER 1: Introduction

We cannot solve our problems with the same thinking we used when we created them.

~Albert Einstein

1.1 Introduction

This thesis aims to provide a conceptualization of program management practices within software development small enterprises (SMMEs) that utilize the Scrum software development approach. This chapter provides an overview of the material covered in the whole thesis. It begins with a background context to agile software environments in general, before providing an overview of the research problem of interrelated connections between software SMMEs, Scrum as a software development approach, and multi-project program management. Thereafter an overview is provided to the research aim and objectives, the research process and the importance of the research. The chapter concludes with a roadmap to the organization of the rest of the thesis.

1.2 Background to the research problem

More often than not, academic and practitioner reports portray Information Technology (IT) projects as plagued with poor success rates. This persistent problem has served as the catalyst for an ongoing search for alternative management approaches that might offer reprieve from this undesirable situation (Ambler, 2008, West and Grant, 2010). One approach that has ascended through the ranks during this quest and demonstrated its potential to offer an alternative means of management, and alleviate some of the tension associated with this brewing IT project crisis (Arell et al., 2012), is agile software development. With its departure from the mentality of strict upfront planning and control mechanisms, in favour of encouraging closer customer involvement and dynamism in software requirements; agile software development has offered a new paradigm for the management of IT projects, that better copes with the challenges of growing project complexity (Highsmith, 2000, Cockburn, 2002) and rapidly changing business needs (Boehm, 2002, Augustine et al., 2005, Conboy, 2009, Barton, 2012).

Scrum is recognized as one of the more widely used contemporary agile project management frameworks (Ambler, 2009, Wang et al., 2012) and it has amassed a significant number of

academics, practitioners and researchers as followers (Lee and Xia, 2010, West and Grant, 2010, Whitaker, 2010). When implemented successfully, Scrum has the potential to offer a relatively simple and adaptable framework for coordinating project-level activities (Schwaber, 2011). Despite the above-mentioned accolades, a review of the literature reveals that there have been a relatively low number of publications that have explored project management practices within an agile software environment at a higher level of abstraction than the individual project level,. Such higher level of abstraction is in some cases referred to as multi-project management and represents an area focused on coordinating multiple, concurrent, competing software development projects. Recent research attempts, such as those by Vähäniitty and Rautiainen (2008b) and Van Waardenburg and Van Vliet (2013), are evidence of growing interest in the multi-project management space in agile software development.

This thesis contends that there is currently a general lack of well-accepted principles and practice guidelines to support multi-project management within the context of agile software development, and endeavours to make a contribution in this emerging research area.

1.3 Research problem

SMMEs are perceived to be important contributors towards economic growth, stability, and job creation in South Africa (Visagie, 1997, Berry et al., 2002, Olawale and Garwe, 2010). At an international level, software development SMMEs appear to make up a significant portion of GDP (Richardson and von Wangenheim, 2007, Taylor et al., 2008), and are also positioned as being strong contributors to the building of a knowledge economy that can promote both international competitiveness and technology innovation (Miettinen et al., 2010). SMMEs are often characterised by small operating budgets, limited external funding, a small workforce, and limited operating resources (Alajoutsijärvi et al., 2000, Saastamoinen and Tukiainen, 2004, Azar et al., 2007). In light of these deficiencies, they typically exist in a high state of dynamism and reflexivity and engage in a constant search to maximise efficiency of production practices and project management practices (Pino et al., 2010b) in order to remain competitive. As a consequence, the characteristics of software development SMMEs call for project management practices that differ from those embodied in the more established and sacrosanct traditional project management frameworks (Fayad et al., 2000).

Project management emerged as a fledgling discipline in the late 1950s (Levin and Green, 2010) and currently the literature posits it as the most prevalent vehicle for business related

actions (PMI, 2014). Typically, project management practices are grouped into major activities such as planning, organizing, staffing, controlling and directing (Kerzner, 2013), whilst project management knowledge areas are sometimes described as containers of practices and include scope, time, cost, quality, human resources, communications, risk and procurement (Schwalbe, 2013).

It is widely recognized that SMMEs find the implementation and upkeep of traditional management frameworks too much of an economic and operational burden (Rising and Janoff, 2000, Habra et al., 2008, Laporte et al., 2008a). Instead, they thrive on creating and sustaining less bureaucratic project environments that, along with other traits, support constant innovation in both product and process (Hoffman et al., 1998), a multi-project nature of operation (Vähäniitty and Rautiainen, 2008b, Miettinen et al., 2010, Lee and Yong, 2013) and the ability to quickly change tack in the face of shifting product and client needs (Rising and Janoff, 2000). In place of traditional project management philosophies, radical alternatives such as those proposed by Denning (2010a) and Martin (2011), appear more aligned with SMME traits such as those listed above.

In the context of software development, Scrum surfaces as a contemporary alternative management approach for software projects (Schwaber and Beedle, 2002, Sutherland, 2010). It subscribes to the principles of agile software development established by the agile manifesto (Agilemanifesto, 2001) and recommends customizable techniques and methodologies for the project space of software development (Scrum.org, 2013). The literature is abundant with studies on Scrum application (Schwaber and Beedle, 2002, Pham and Pham, 2012), process improvement using Scrum (Kim, 2007, Sutherland et al., 2008, Pino et al., 2010b), and team organization for Scrum (Beedle et al., 1999, Rising and Janoff, 2000).

It is the view of this thesis that isolated project level practices are well publicised within the discourse of Scrum. A myriad of journals, blogs, websites, technical and practitioner experience reports, and books disseminate experiences of the technicalities and practices within a Scrum software project. At the same time, in a growing, competitive economy like South Africa, it is becoming an increasing relevant requirement for companies to be able to manage multiple projects simultaneously (Jonas, 2010), and this need is exaggerated in the limited resource environments typical of SMMEs. Multi-project development has been an active area of research ever since it peaked in the 1990s (Levin and Green, 2010). Related topics, such as scaling Scrum to enterprise level management, have been gathering

momentum over the past decade (Abrahamsson et al., 2003, Mahnic and Drnovscek, 2005, Dingsøyr et al., 2006, Cervone, 2010). Despite this shift, the research area of program management appears largely unsettled (Thiry, 2010) and a concise and well accepted body of knowledge remains elusive. Literature concerning Scrum in the discourse of program management practices remains under theorized. It is the argument of this thesis that well-established and well-accepted program management theory is still a long way off due to insufficient academic contributions in this area. While Scrum practices manifest in software companies of varying sizes and focal areas, SMMEs have been singled out for this thesis because there appears to be a gap in knowledge pertaining to multi-project management practiced in this category of enterprises. This line of argument has led to the following research question:

How is program management carried out in software SMMEs practicing Scrum?

1.4 Research aim and objectives

Theoretical propositions help to focus the investigative effort (Runeson et al., 2012) and guide the fieldwork of a research undertaking (Yin, 2009, Remenyi, 2012). The major proposition made in this study is that the union of the Scrum software approach and characteristics of software SMMEs, spawns a new and relatively unknown pattern of program management. Hence the research aim is stated as:

> Elucidate the nature of the interfaces and interactions between program management practices and software SMMEs practicing Scrum.

Stated differently, the main aim of this study is to conceptualize a view of the interfaces and interactions between defining characteristics of software SMMEs, Scrum software development approach, and the resultant practices of program management. In order to achieve this overarching objective, it is incumbent to first, establish the characteristics of software development SMMEs; second, in light of these characteristics, to understand how Scrum is utilized as a vehicle for management of software projects in the context of software SMMEs; and finally, to further elucidate program management, its knowledge areas, and its associated practices.

The study's research objectives (RO) can thus be listed as follows:

RO1: Garner knowledge on software development SMMEs with emphasis on their distinctive characteristics
RO2: Explicate the Scrum software development approach including major practices, artifacts and roles
RO3: Synthesize the knowledge areas and practices of current program management discourse

Figure 1 provides an illustration of the three areas that form the core of this thesis:

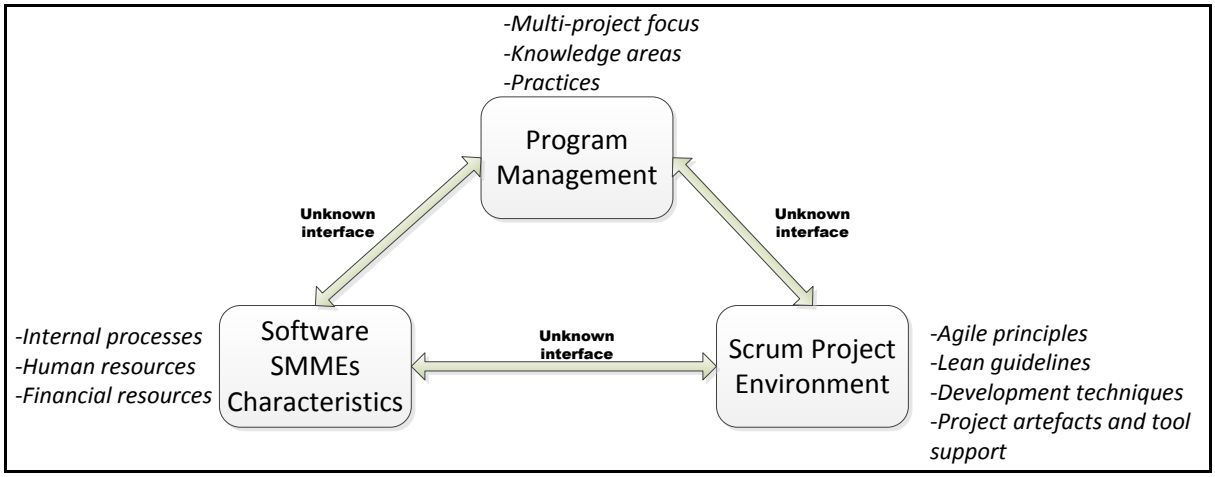


Figure 1: Disconnected areas that form the core of this thesis.
Source: Researcher's own construction

The literature review and the research objectives facilitated the distillation of the research questions, and these are introduced in Chapter Three.

1.5 Research process

A qualitative, interpretive approach was adopted for this study. A qualitative approach typically requires immersion of the researcher into the environment under investigation, allowing him/her to gain in-depth insights into phenomena (Gregor, 2006, Walsham, 2006a). This approach was deemed most suitable to investigate the interplay between Scrum, SMME characteristics, and program management, as it allows for the researcher to gradually improve relationships with a few SMME software project teams while employing techniques better suited to the different formats and sources of data being collected (Lee and Baskerville, 2003). Interpretive research is frequently elected for studies that aim to establish an understanding of phenomena derived from people's understanding of the situation (Myers, 2013). The interpretive stance was elected for this study in order to satisfy the need for

appreciating and consolidating different participant perspectives elicited during the fieldwork. Furthermore, data collection within a real life setting, as is the case in this study, presides over artificially created environments. This is often contended as a key tenet of interpretivism (Smith, 2006).

The case study methodology suited this study because it permitted more in-depth investigations of the complex issues investigated within their natural settings in this study (Yin, 2012). The case study methodology utilized in this study best conforms to the descriptive category (Tobin, 2010, Yin, 2012). This choice of descriptive category was motivated by the need to conceptualize the interface between Scrum, SMME characteristics, and program management. In other words, the aim of this study is to generate theory and not apply or test existing theory, and thus the study endeavour is best supported by the descriptive case study methodology (Rule and Vaughn, 2011, Runeson et al., 2012). Two software SMMEs volunteered participation in this study and they formed the case study sites.

Fieldwork lasted for eight months in total, roughly four months per case study site. The duration was dictated mainly by leave permitted to the researcher; and,this was sufficient time because not much new data was being generated towards the end of the fieldwork. Furthermore, the researcher was not completely alienated from Scrum due to teaching courses on the subject matter.

Data collection primarily employed ethnographic techniques of observation, interviews, and also included limited participation in certain project activities. Principles and guidelines for ethnography were adhered to, such as those proposed by Forsythe (1999), Silverman (2010), and Creswell (2013). Some of these guidelines included unobtrusive observation that does not disturb the participants from their daily practices, and unstructured interviews that reduce researcher bias and avoid *shepherding* responses. Limited participation was helpful in allowing for more in-depth exploration of practices and elucidating the nature of influences upon them. The researcher did not impose this request; instead he participated only when invited. Data generated by fieldwork was transcribed and catalogued using a simple indexing technique. This was necessary for the efficient management of the large amount of detailed fieldwork notes and lengthy interview responses. Project artifacts were carefully monitored in their usage by project teams as was their subsequent influence on practices.

The order in which ethnographic techniques were employed is as follows:

a. Observation was conducted for the majority of fieldwork duration
b. Participation in activities and monitoring of project artifacts was done throughout
c. Interviews were done at the end of the fieldwork at each case study site

Each case study site was treated as stand-alone, and the findings at each site were consolidated towards the end of the study. Activity Theory (AT) formed the major analysis lens for data collected at each case study site. Concept analysis and thematic analysis was implemented later on to refine findings and generate the eventual contribution of this study.

1.6 Importance of the research

In a special issue of EJIS, Abrahamsson et al. (2009) point out that agile practices are currently being used well beyond their intended area of applicability and this motivates the need for supporting evidence, especially in the areas of agile methods, practices, and innovation. These authors stress that "one of the most pressing issues is the need to develop a better understanding of the implementation of agile at the organizational level" (Abrahamsson et al., 2009). A closer look at the literature reveals that:

> "Underneath the surface of general descriptions and recommendations concerning teams, there is indeed fairly little critical and original theorizing on the collaborative work and associated cognitive and communicative processes within and between teams in real organizational contexts" (Engeström, 2008).

There is thus an acute awareness, among researchers and practitioners, of the general lack of well-grounded knowledge on the nature of practices beyond the singular team and singular project level.

The literature is abundant on practice of Scrum at the single team and individual project level. At the time of writing Google Scholar keyword combinations like *agile and management* or *agile and project team*, returned over 200 000 articles. However, it is the proposition of this thesis that research contribution in the program management space requires an increase in both rigor and validity. Despite the researcher's best efforts, no studies were found that concentrate specifically on trying to conceptualize program management practices within the context of an SMME practicing Scrum. This thesis represents a committed effort to address this gap.

This research study can thus provide a valuable contribution to the field by elucidating the interfaces between the three core areas of SMMEs, Scrum and management practices, and by providing a perspective of the relationships of influence between Scrum and program

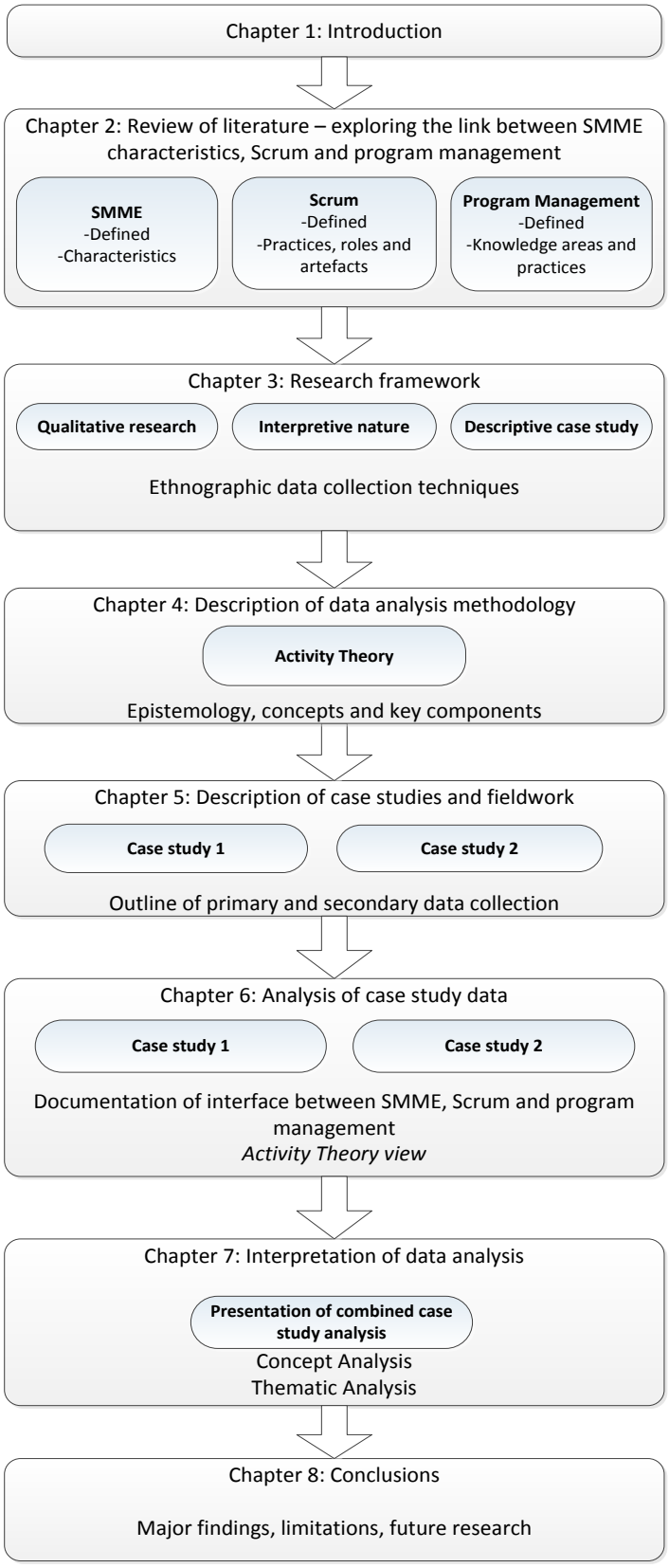
management practices embedded within software SMMEs. A conceptualization of this interface could make an additional contribution by providing clues towards improved adoption strategies, and by offering some insights into requirements for harnessing agile within management frameworks. In turn, it is hoped that the developed theory could serve as a starting point for further research.

1.7 Organization of this thesis

This thesis is organized into seven chapters. Chapter One has provided an introduction to the study as a whole. Chapter Two reviews and interprets current literature in an attempt to establish software SMME characteristics, the best practices of Scrum software development approach, and program management practices. Chapter Three introduces the research questions and describes the research framework as well as the underlying philosophy, strategy, methodology, and data collection techniques. Chapter Four focuses on the data analysis lens namely, Activity Theory (AT). Chapter Five provides a description of the fieldwork and details the two SMMEs that volunteered and collaborated as case studies. Chapter Six provides an analysis and elaboration of the data analysis collected at each case study. Chapter Seven provides an interpretation of the data analysis and summarises and presents the results. Chapter Eight concludes the thesis with a summary of theoretical contribution, limitations of the study, and future research suggestions.

Figure 2, which is illustrated across two pages below, will serve as a roadmap to guide the reader through the thesis and provide a concise overview of the whole study.

Figure 2: Roadmap of this thesis.
Source: Researcher's own construction



CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of literature relevant to each of the demarcated research areas; these being, the characteristics of SMME management practices, the agile software development and Scrum framework, and program management knowledge areas and practices. After briefly introducing these three research areas of interest in section 2.2, a more detailed review of literature is provided for each area individually.

The first section (2.3) provides a definition of an SMME as subscribed to by this thesis; reviews perspectives of SMME economic contributions; and portrays a synthesis of SMME management practice characteristics. The second section (2.4) proffers a definition for agile software development and Scrum; portrays a conceptual link between agile and Lean principles; synthesizes the key roles, practices, artifacts and metrics utilized in Scrum software projects; showcases current pragmatic perceptions of agile; and explores current points of view on management thinking underpinning agile. The third section (2.5) defines program management; juxtaposes program management with the related areas of project and portfolio management; explores perceptions on the underpinning management philosophy; provides a synthesis of knowledge areas and practices and a view of the life cycle model; and presents literature showcasing contemporary areas of research, including some persisting challenges. The chapter ends with a summary of the findings of the literature review.

The aim of the chapter is to explore recurring themes and contentious issues in the three selected research areas, in order to bring to light sufficient insights to facilitate a theoretical conceptualization of how these three areas interface.

2.2 Broad overview of the research areas covered

This thesis speculates a current disconnect in the literature between the study's three core areas of management characteristics of software SMMEs, the Scrum software development approach, and program management practices. Before embarking on a detailed investigation of the literature in each core research area, it is worth providing a very broadstrokes picture here of the management characteristics of software SMMEs, the Scrum software development approach and program management practices, before delving into the detail.

As regards the research area of SMME management characteristics, the two case study sites selected for this study were software SMMEs, both of which have been developing software for less than ten years. In line with this, section 2.3 will showcase a synthesis of literature that provides a perspective of knowledge areas and characteristics of management practices of typical software SMMEs. In so doing, it will be demonstrated how these characteristics are honed from the need to remain profitable in light of deficiencies in resources and manpower that appear almost distinctive to small companies. A synthesis of these characteristics is considered important to this thesis due to their direct influence on the shape of Scrum and on the nature of program management practices.

In the second research area of the Scrum software development approach (covered in section 2.4), the literature indicates that agile software development has been gaining followers over the past two decades, primarily because of its claims that its use improves time to market and leads to a software product that more closely matches client needs. (This growth in adoption is evident in the fact that of the eight software development companies that were approached to volunteer as case studies for this thesis, two were practicing Scrum and five were testing Scrum within pilot projects). The literature indicates a proliferation of research focus on the areas of agile adoption and implementation at the project level. In addition, current research is starting to show an interest in how to better understand, and how best to shape, management practices at a higher organizational level such that software companies are better equipped to accommodate agile.

The third area of interest, program management, is covered in section 2.5. Program management typically embodies planning and controlling practices within a multi-project context; in other words, it embodies the organization and co-ordination of a group of concurrent projects. Thus, the primary objective of program management is to harmonize the coexistence of competing projects. Practices seek to cater for functions such as resource sharing, planning, conflict resolution, and maintaining operational infrastructure. Professional bodies, like Project Management Institute (PMI) and Office of Government Commerce (OGC), publish and update guidelines for best practice in program management. However, some practitioners and researchers are now beginning to question the effectiveness of the more rationalist, positivist approaches to program management campaigned by such professional bodies. These practitioners and researchers seem more sympathetic towards an emerging paradigm of management thinking which explores alternate practices in line with the demands of modern software project environments. Questions that are being asked

include the following: Is this emerging paradigm of management thinking a suitable ecosystem for agile? Is it pliable within SMMEs? How is this format of management operationalized via associated practices in the context of agile and SMMEs?

The sections that follow delve deeper into each of the above-mentioned core areas in search of answers to these questions. Software SMME management characteristics are covered first, followed by Scrum, before dealing with program management.

2.3 Characteristics of management practices of software development SMMEs: A view

This section aims to synthesize the characteristics of management practices in SMMEs. But before doing so, a definition of software SMMEs is provided and the economic importance of SMMEs is explored.

2.3.1 A definition of SMMEs

The South African government national Small Business Act (No. 26 of 2003) classifies a small business as follows:

> "…small enterprise means a separate and distinct business entity, together with its branches or subsidiaries, if any, including co-operative enterprises and non-governmental organisations, managed by one owner or more predominantly carried on in any sector or subsector of the economy mentioned in column 1 of the Schedule and which can be classified as a micro-, a very small, a small or a medium enterprise" (Department of Trade and Industry 2003: 2).

Small businesses are more commonly referred to as Small to Medium Sized Enterprises (SMMEs). As pointed out by the above definition and supported by Olawale and Garwe (2010) and Olugbara and Ndhlovu (2014), SMMEs are typically started by and run by an individual in the role of *owner-manager.* In some cases there may be other senior roles such as directors and managers, but this is largely dictated by the type of SMME.

The Department of Trade and Industry (DTI) provides a classification scheme for SMMEs operating in Southern Africa (DTI, 2008). This classification requires the examination of criteria mainly relating to the number of employees, annual turnover and gross asset value, as well as the economic sector within which the SMME operates; for example, agriculture or manufacturing. A recent MICTSeta report indicates that there are almost 9400 IT businesses (MICTSeta, 2013), which signifies a considerably sized sector. Despite this statistic, DTI does not allocate a separate economic sector for IT in its economic classification. Instead, IT is positioned within the Finance and Business sector. Adhering to the thresholds for this

sector, SMMEs can be described as micro, very small, small or medium. Table 1 compares these thresholds:

Table 1: Comparison of SMME thresholds in the finance and business sectors (Source: DTI 2008: 3)

Classification	Number of permanent employees	Annual turnover(R millions)	Assets value(R millions)
Micro	5	0.2	0.1
Very small	20	3	0.5
Small	50	13	3
Medium	200	26	5

The two case study software SMMEs will be classified using these thresholds in order to better align the findings of this study with other studies involving South African SMMEs.

2.3.2 Perceived economic importance of SMMEs

As early as the 1990s, SMMEs have been positioned as key components in government policies for economic development, job creation and poverty alleviation (Berry et al., 2002, DTI, 2008, Nicolaides, 2011, Olugbara and Ndhlovu, 2014). Recent statistics indicate that SMMEs in South Africa are responsible for the employment of 56% of the employable population (Olawale and Garwe, 2010) and that SMMEs contribute significantly towards job creation and GDP (Nicolaides, 2011, Mujinga, 2013). Statistics show an increasing growth trend in the number of people being employed by SMMEs across all sectors (MICTSeta, 2013).

The importance of SMMEs is recognized in other economies too. Olugbara and Ndhlovu (2014) point out that in most overseas economies with low per capita income, SMMEs constitute a significant proportion of businesses that contribute towards the economy. SMMEs typically have the ability to produce innovative products and services while being flexible to their client demands (Laporte et al., 2008b, Wadood and Shamsuddin, 2012). Some literature appears more focused on the importance of software development SMMEs. Richardson and von Wangenheim (2007) claim that as many as 85% of software companies are classified as small companies in nations around the world. Recent statistics indicate that 94% of all software development companies in the EU are of the small to medium category, and have fewer than 50 employees (Pino et al., 2010a). The same authors indicate that 75%

of software companies in Latin America fit into the same category. These are significant proportions and highlight that SMMEs represent an area of research and practice that transcends international boundaries.

2.3.3 The challenge of lack of statistics on South African software SMMEs

Government intervention strategies instilled to support SMMEs have recently been brought under scrutiny (PMG, 2012). This is indicative of interest in the success and sustainability of SMMEs in South Africa, a trend started and supported almost two decades ago by the governments of the UK (Hoffman et al., 1998) and the US (Richardson and von Wangenheim, 2007). Despite such initiatives, DTI contends that in comparison to their peers in overseas economies, South African SMMEs are less dynamic, with the majority of them still in the *baby business* phases (DTI, 2008). DTI further stresses a mismatch in size threshold when compared to developed countries; for example, a SMME that might be classified as medium or small in the US and EU is considered large in South Africa.

DTI does not provide a separate sector classification for SMMEs in the area of IT, which would include software development. This does not align well with MICTSeta's classification of the South African IT sector. MICTSeta is tasked with identifying and addressing skills development and training needs within the South African economic sectors of Advertising, Film and Electronic Media, Electronics, Information Technology, and Telecommunications. The following statement was made in a recent MICTSeta report:

> "The combination of Media with ICT into an economic sector is unique to South Africa and does not follow international conventions of industry classifications and this makes it complex to conduct research at MICT level" (MICTSeta 2013: 14).

There are limited academic and government studies focusing on South African software SMMEs. To the best of this researcher's knowledge, no recent statistics are available for South African SMMEs in the segment of software development. A key contributing factor could be that major government departments, like DTI and South African Revenue Service (SARS) do not provide classifications for software SMMEs or separate IT sub-sectors within the economy (MICTSeta, 2013). As a result, SMMEs at the time of writing this thesis, are grouped into other classifications such as Finance and Business or Other Services.

In summary, the above notions express graves concern that the South African IT sector has not been more adequately classified at the level of national data gathering, and this serious hampers research efforts in the area of software development. Among other challenges, it

makes it difficult to compare South African SMMEs with their international counterparts, which may lead to their exclusion from wider research and academic debates. Comments of this nature point to the need for more academic advocacy on the issue of classification, and a wider research focus and more contributions in the discourse of South African SMMEs in general and SMMEs operating within IT arenas in particular. Despite these limitations, this study conforms to the definition of SMMEs as outlined by the DTI and the small business act. This is deemed necessary in order to align the discussion on SMMEs with other research efforts on SMMEs conducted in South Africa.

2.3.4 A synthesis of SMME management knowledge areas and their characteristics

SMMEs employ fewer people and generally take on fewer concurrent projects when compared to their counterparts in big firms or large corporates. Naturally, their internal processes and operations seldom resemble those found in bigger companies (Hoffman et al., 1998, Rising and Janoff, 2000, Wadood and Shamsuddin, 2012, Olugbara and Ndhlovu, 2014) and often require constant innovation of both product and process (Saastamoinen and Tukiainen, 2004, Laporte et al., 2008a), in combination with aggressive resource management (Richardson and von Wangenheim, 2007). Innovation in product or service offering and management practices represents a key survival mechanism for software SMMEs given their usual resource and manpower constraints (Rycroft and Kash, 2004, McAvoy and Butler, 2009) and their lower tolerance for project failure (Pino et al., 2010a). As a result, SMMEs must strive to ensure sustained innovation not only in processes for product development but also in overarching management practices as well (Laporte et al., 2008a, Pino et al., 2010a). Unfortunately, the literature finds that SMMEs are less likely to recover from failed experimentation (Azar et al., 2007, Taylor et al., 2008).

Taken together, these findings highlight the key drivers that often define the nature of SMMEs' management practices. A point of view on these, synthesized from the literature above and from Coleman and O'Connor (2008), Laporte et al. (2008a), Taylor et al. (2008), Pino et al. (2010a), Pino et al. (2010b), Higgins and Mirza (2012), Hurtado et al. (2013), Mujinga (2013) and (Olugbara and Ndhlovu, 2014), is presented below. A process of thematic analysis was applied to achieve the presented results. There are variations to the practical application of thematic analysis; for examples see Oliveira et al. (2013). For this section, the variation applied is one of continually organizing and reorganizing similar themes in the literature until a relevant set of similar concepts is generated (Fereday and

Muir-Cochrane, 2008, Vaismoradi et al., 2013). By applying a thematic analysis approach to relevant literature, three major knowledge areas were identified from the literature. These areas form the containers for characteristics, that are illustrated in Table 2.

Table 2: SMME management practices: Knowledge areas and characteristics

Knowledge Area	Characteristics
Day-to-day operations; including planning, controlling, communicating, reporting and quality management	• Responsive and dynamic in light of gaining competitive advantage • Mainly informal coordination and control mechanisms • Direct and frequent communication • Mostly aim to develop software solutions within a niche area • Lightweight process methodologies • Preference for shorter delivery cycles
Human Resources	• Flat organizational structure with highly competent employees • Creativity and innovation are encouraged • Expected to perform in more than one area of expertise • Close interaction between employees
Financial Resources	• Limited finances • Low tolerance for failed projects

This section does not pretend to offer a concise nor a definitive list of SMME management knowledge areas or characteristics. However, it does serve as a starting point and provides clues to the nature of practices that might be encountered during the investigative fieldwork. Other studies utilizing software SMMEs as case studies have found similar characteristics for example, Richardson and von Wangenheim (2007), Wang (2007) Cho (2010) and Van Waardenburg and Van Vliet (2013). This serves as a degree of validation of SMME characteristics in Table 2. The next section deepens the discussion on agile software development and Scrum, which comprises the second primary research area of this thesis.

2.4 Overview of the Scrum software development approach

Just over a decade ago, Boehm (2002) and Highsmith (2002a) prophesized agile practices as the future dominant approach for software development, raising the question of whether or not these authors predicted correctly. This section provides a definition of agile software development and Scrum. It then explores the link between the principles of agile software development and Lean; synthesizes Scrum roles, practices, artifacts and metrics; provides a

view of the pragmatic effectiveness of Scrum, and reports on an investigation into the implementation of Scrum at higher organizational management levels.

2.4.1 A definition of agile software development approaches

Software systems are usually crafted by following a process model, and the interrelated activities of software development are completed in accordance with the dictum of the elected process model. Software process models are usually positioned on a scale with polar opposites of classical (plan-driven) or agile. Classical process models strictly adhere to a predetermined format and sequence of activities. On the other end of the scale, agile process models encourage a degree of process model tailoring, provided that the underlying agile philosophical principles and guidelines remain intact. The latter process models are often perceived as vehicles for software project level management.

The literature shows that the classic software process model was the first formal approach to building software systems (Sliger and Broderick, 2008, van Vliet, 2008, Braude and Bernstein, 2011, Sommerville, 2011) in an era when engineering philosophies were being inducted into software development. This classic model is often seen as an early attempt to provide a structured approach to the increasingly complex task of developing software systems (Pfleeger and Atlee, 2010). One of the earliest, and possibly most widely adopted, classical process model is the Waterfall model (Royce, 1970). Boehm and Turner (2005), Dyba (2005), and Misra, Kumar, and Kumar (2009) opine that classical software project management can be comfortably placed in the rationalistic (or prescriptive) management paradigm which assumes task characteristics of standardization, controllability and predictability. Classical software development projects typically require an early, detailed plan and progression through predefined sequential phases (Pfleeger and Atlee, 2010, Pressman, 2010), with monitoring and control mechanisms inserted at various stages of the project. Hence, they are sometimes referred to as a plan-driven approach (Cockburn, 2002, Schach, 2011).

Publications by authors such as Pfleeger and Atlee (2010), Augustine et al. (2005), Pikkarainen et al. (2008) and Cho, Huff and Olsen (2011), reminisce that agile software development was pioneered during the 1990s by a group of practitioners who felt that the strict procedure of classical approaches was not well suited for operating in what they called turbulent business environments. *Turbulent business environment* is a popular catch phrase in agile circles that is used to describe requirements uncertainty and volatility. It was felt that

the needs of businesses at the turn of the millennium required software approaches that could accommodate this so called turbulence.

Agile software development does not wholeheartedly subscribe to strict structure in its process model or to detailed upfront planning. Instead, agile perceives the business needs of a client as emergent and dynamic, and believes the associated software development process should have the tactility to withstand the related pressures as the project progresses. In light of uncovering these dynamic business properties, agile software development typically exhibits traits like reduced development time for working software demonstrations, and intimate interaction between the project team and the client.

The literature frequently labels agile development process models as an evolutionary approach (Nerur et al., 2005), in that naturally evolving business requirements are tolerated during production of the software. Agile models are thus sometimes referred to as incremental and iterative process models (Schach, 2011, Sommerville, 2011) because they exhibit an 'inspect and adapt' philosophy towards the conflicting and unpredictable demands of software projects (Williams and Cockburn, 2003). There are several variations on the agile theme, but most agile process models emphasize the following core practices, as suggested by Cockburn (2007), Fairley (2009), Conboy, Wang and Fitzgerald (2009) and Schwaber and Sutherland (2011):

a. Agile process models continuously involve the client as an active change agent and a contributing member throughout the project;
b. They develop test cases and test scenarios before implementing the next version of the product;
c. They provide frequent demonstrations of each version of the evolving software to the client;
d. There is constant elicitation and refinement of business requirement(s);
e. There are short periodic delivery cycles into the operational environment.

The following summation provides a synthesis of some of the core practices:

> "The development of a system in part as inspired by the customer and the continuous integration of the parts following planned repeated modifications which the developer, team and techniques emerge into a winning system through the use of simple concepts such as flexibility, simplicity, trust and choosing minimal overhead route that will lead to working code" (Mnkandla, 2010).

Furthermore, agile software practitioners believe that agile provides an approach for software teams to provide results more effectively when compared to command and control approaches (Pham and Pham, 2012).

In summary, agile exists in many permutations. Some of the more popular include Extreme Programming, Feature-Driven Development, Dynamic Systems Development, and Crystal. Scrum is another popular variation of agile software development. Both software SMMEs in this study were practicing a variation of Scrum. The next subsection defines and explains Scrum in more detail.

2.4.2 A definition of Scrum

Scrum can be defined as a framework of best practices based on the principles of agile software development (Schwaber and Beedle, 2002, Sutherland, 2005, Sutherland, 2010, Schwaber, 2011), and is not intended to be a strict, prescriptive formal methodology. Nonetheless, Kelly (2013) opines that Scrum is usually positioned as a prescriptive methodology during the early adoption stages and only when the team becomes comfortable with tailoring its processes does it become more akin to a framework. Due to its nature of continuous tailoring, enshrined in agile principles like *people over process*, Scrum is often implemented and practiced in some variation of the original version first proposed by Sutherland in 1995 (Schwaber and Sutherland, 2011).

Before a detailed deliberation of Scrum practices is provided an exploration of its underpinning in Lean and agile principles is necessary. This is done in an attempt to lay a foundation which may help to better understand Scrum practices.

2.4.3 A comparison of Lean and agile principles

The agile manifesto (Agilemanifesto, 2001) was created, and is maintained, primarily by practitioners and academics who believe that agile software development is the *only* approach for modern software projects. This manifesto seeks to spread the philosophies of agile development in the form of teaching the principles and values which underpin most modern agile process models. While investigating agile principles and value propositions, it became evident that the literature meandered towards principles of Lean manufacturing. The literature indicates that Lean is a particularly vast and volatile area of research and this section will not attempt to provide either a concise or a complete elaboration thereof. Instead, the aim of this section is to provide just enough background and detail of Lean to allow for better understanding of the foundations of agile practices.

Lean represents a departure from the classical school of management theories which were pioneered by Max Weber's bureaucratic management of the 1920s, Frederick Taylor's

scientific management in the early 1900s, and administrative management by Henri Fayol in 1880 (Grant et al., 2010). Instead, Lean incorporates some characteristics of management approaches first utilized in the Japanese economy and appears to be guided by principles such as: 'reduce wastage along the production line', 'only provide products which add value for the client business', 'autonomous teamwork', 'reduced specialization of work' and the 'continuous improvement of the production process' (Grant et al., 2010, Panizzolo et al., 2012, Bosch et al., 2013). These principles of Lean were honed with the aim to drive manufacturing closer to an improved production process (Nordin et al., 2012). The key principles of Lean, as outlined by Womack and Daniel (2003), are as follows:

a. Determine the meaning of value from the perspective of the customer
b. Analyze all the steps in the production process and eliminate those which do not add value
c. Ensure the value creating tasks of the process provide the customer with the product in constant and even delivery cycles
d. Involve the customer in a *pull* process whereby nothing is produced unless the customer requires or requests it. (This differs from the entire product being built to completion and then *pushed* onto the customer)
e. In the pursuit for perfection these steps are on-going and never ending as perfection is an ideal that will never be 100% satisfied

When juxtaposed, the principles of Lean can be traced to principles of agile. Table 3 illustrates this relationship. The content of the table is derived from a short thematic analysis of literature available from professional body websites such as Agilemanifesto (2001) and LEI (2009).

Table 3: Comparison of Lean and agile principles

Concept	Lean Principles	Agile Principles
Value	Value is measured from the perspective of the client	Provide value for the client through continuous delivery of valuable software
Process refinement	Eliminate process steps that do not add value	Simplicity – maximise the amount of work not done
Delivery schedule	Constant delivery of value to the client	Frequently deliver working software
Client involvement	Produce only what is required by the client	Business people and developers must work closely together
Process Improvement	Continuous improvement of the process where perfection is an ideal not an end point	At regular intervals team reflects upon their processes and asks how to become more productive

2.4.4 On the nature of agile practices

The tabular comparison above endeavours to highlight the link between the underpinning philosophies of Lean and agile software development. Agile practices such as product backlog planning, daily stand-up meetings, retrospective meetings and progress measurements, satisfy many of the principles of Lean. There is ample literature that juxtaposes Lean principles and agile software development practices, and good discussions of this can be found in Hoebeke (2000), Ambler (2009), Poppendieck and Poppendieck (2009), Jonsson (2012), and Wang et al. (2012). Kelly (2013) notes: "I sometimes think of Lean as a meta-model: while XP and Scrum tell you what to do, Lean tells you how to think so that you may decide what to do."

It becomes even more evident from this comparison that agile principles are honed from the relatively more established and more mature Lean principles that were first adopted in manufacturing. In turn, agile practices and their supporting artifacts represent a means of operationalizing the value proposition denoted by agile principles. Figure 3 provides a representation of this hierarchy along with a few examples:

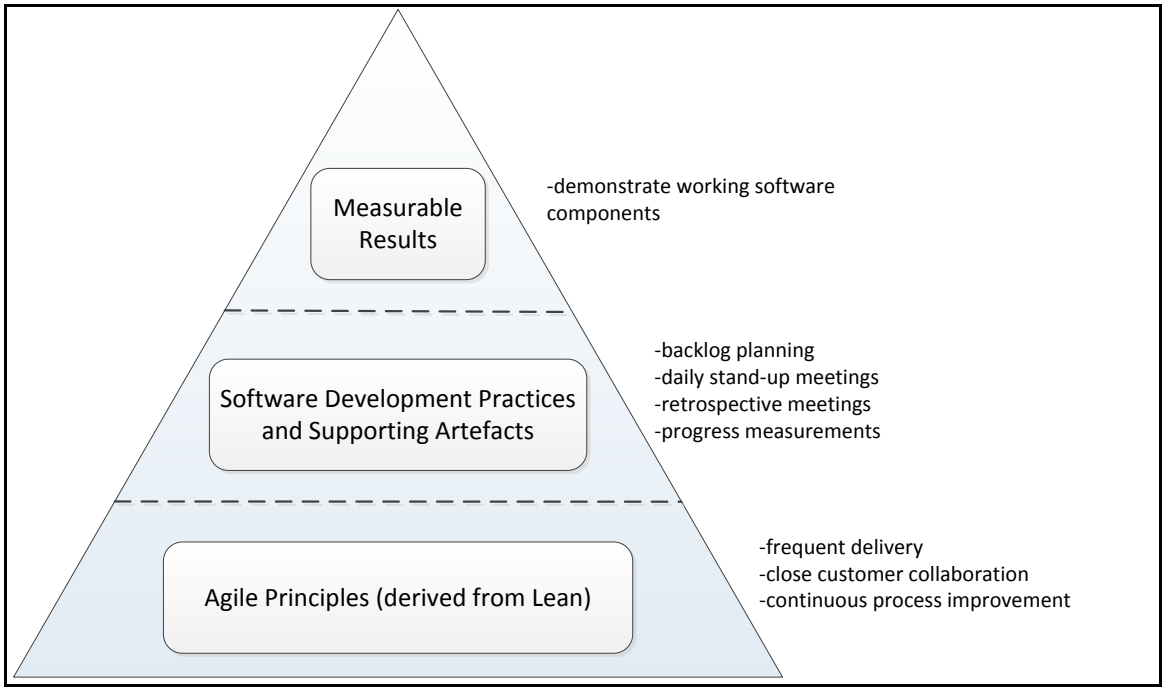


Figure 3: A hierarchical view of principles, practices and results.
Source: Researcher's own construction

Figure 3 is crafted to highlight the link between Lean principles and agile practices and hence to offer a perspective of their nature. An understanding Scrum practices forms a major

component of this thesis. It was necessary to excavate deeply into Scrum's philosophical underpinnings and value drivers to better observe them in their pragmatic setting; that is, to better understand them in practice during the fieldwork, as well as to improve the rigor of the findings of this thesis. For example, literature points to agile project artefacts such as, product backlogs and Kanban which are in place to improve communication of project status. Frequent communication is a principle of agile, which derives from Lean principles. This results in a measurable output such as, client software requirements becoming clearer.

Having provided a perspective on Scrum's underpinnings, the next subsection takes a closer look at Scrum practices, artifacts, metrics and roles.

2.4.5 A brief description of Scrum roles, practices, and band metrics

This section provides a description of the generic roles, practices, metrics and artifacts found in typical Scrum software development environments. The information that follows is derived primarily from Pham and Pham (2012), Pressman (2010), Rising and Janoff (2000), Schwaber and Beedle (2002), Schwaber and Sutherland (2011), Sutherland (2010), and van Vliet (2008). Descriptions may also be found in Rising and Janoff (2000), Lindvall et al. (2002), Mahnic and Drnovscek (2005), Hoda et al. (2008), Moe et al. (2010), and Cervone (2010). Once again, a thematic analysis approach was utilized. The reader unfamiliar with Scrum is advised to refer to the tables presented in the appendix that describe Scrum roles, practices, and band metrics. The paragraphs below put both human and non-human actors described in the appendix in theatre to present a rendition of a typical Scrum software project.

Scrum views and organizes software development project work in iterations called *sprints*. With subsequent iterations the software product more closely resembles the requirements initially requested and negotiated by the client. The typical Scrum project begins with documentation of requirements for the client software product. The *Product Owner* is either the client business owner or representative tasked with the dual role of guardian and knowledge bearer of business requirements. These requirements are arranged in order of importance in an artifact commonly referred to as a *product backlog*. The Product Owner is coached on how to create the product backlog. Individual requirements are known as *user stories*. Usually, senior developers (often with job titles such as systems architect or technical director) utilize this product backlog to envision and design the software architecture. The analogy of an architect's plan for a building is a suitable metaphor to describe the purpose of software architecture. By closely adhering to the priority assigned to each user story,

developers perform *sprint planning* which is a plan of the sequence for developing and delivering the user stories. Each user story is further decomposed into smaller tasks which are coded by the developers during the course of the sprint.

A *sprint* is a period of time during which user stories are coded. A typical sprint lasts four to six weeks and the number of sprints is dependent on the complexity of the project, size, and capability of the development team. There can be any number of sprints, but the final due date for the entire software project is fixed. Each sprint will take on a batch of user stories at the onset and towards the end the objective is to ensure working software which provides the business requirement depicted in each user story. Working software is demonstrated to the client at the end of every sprint with the aim of eliciting feedback on functionality and usage. This is called a *sprint review*.

A typical project team consists of 6-8 developers. Everyday a stand-up meeting is held which is generally referred to as *Scrum* or a *daily stand-up meeting*. The purpose of the daily stand-up meeting is to report on completed work, state work in progress and identify any challenges being experienced within the project. The primary artifacts are the *burndown* and *velocity charts*. *Kanban boards* may also be utilized as a centre piece for the daily stand-up meeting. These artifacts provide all levels of team members, the *Scrum Master*, and management with a finger on the pulse type mechanism for that project. Based on the situation portrayed each day, the Scrum Master in collaboration with the developers may devise and implement short-term project interventions. A *sprint retrospective* is a ceremony held at the end of each sprint. This practice allows a forum in which the project team questions their modus operandi in light of possible improvement. Interventions are usually inserted to attempt process improvement. Figure 4 visually describes this process.

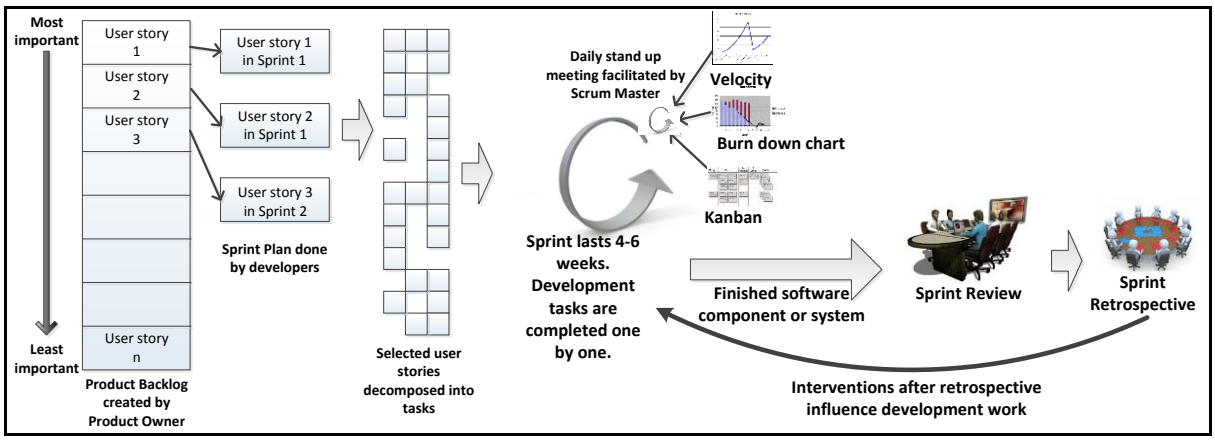


Figure 4: Scrum practices, artifacts and roles in concert.
Source: Researcher's own construction

Having presented the core features and process of Scrum, the next subsection reviews literature on perceptions of adoption and implementation of Scrum.

2.4.6 A view of the effectiveness of scrum in software development environments

> "Agile adoption is a reality. Organizations across all industries are increasingly adopting agile principles, and software engineers and other project team members are picking up agile techniques" (West and Grant, 2010).

A recent survey involving 4770 developers from 91 countries concluded that 90% of the respondents worked in companies that were using agile in at least 50% of their projects (VersionOne, 2010). After empirical field work involving almost 1300 IT professionals, Forrester Research (West and Grant, 2010) declared that agile software development methods are popular among development teams, and even surpass classical methods by a small margin. Interestingly, the study reports that the majority of IT professionals reflected no formal approach or methodological adoption to software development. There are currently a large number of studies taking place within the realm of agile adoption (Dingsøyra et al., 2012) and this indicates impetus towards maturity of the field. Major US-based companies such as Ford, Boeing, FBI, NASA and US Department of defence (Barton, 2012) as well as Yahoo, Microsoft and Adobe (Sutherland, 2005) have adopted agile to some degree at the project level. Baskerville, Pries-Heje and Madsen (2010) conclude that by 2008 agile methodologies had become the dominant approaches in the US and certain EU countries.

Cervone (2010) and Abrahamsson et al. (2003) argue that Scrum is a simple to understand approach for project management that reduces administrative work done by the development team and hence liberates them to concentrate on the core task of software production. Other claims about Scrum's benefits include statements that Scrum is "a viable alternative to face the current software crisis in large projects" (Ktata and Lévesque, 2009a) and that "Scrum has attracted significant attention amongst software practitioners during the last five years" (Marcal et al., 2007). Greening (2010) and Mahnic and Drnovscek (2005) report that implementation of Scrum considerably improves project-based profit. Pikkarainen et al. (2008) acknowledge that Scrum improves communication between the project team and stakeholders. Yahoo! implemented Scrum over a two year period and claims that 150 development teams use Scrum for project work, of which 74% feel that it improves productivity (Benefield, 2008). Elaborating further, Benefield highlights that the following adoption process was followed:

> "We started with Scrum, using its lightweight framework to create highly collaborative self-organizing teams that could effectively deliver products to market. Next we started to add in Agile engineering practices and Lean fundamentals to deliver greater business value and reduce organizational waste" (Benefield, 2008).

The literature hints that correct implementation could see the realization of advantages such as the resultant product being a closer match to requirements, shorter delivery times, and the ability to better manage the creation of more complex software systems (Beck, 1999, Ambler, 2002, Boehm, 2002, Cockburn, 2007, Barton, 2012). There are practitioner reports that claim to have realized an improvement in their software development process upon shifting away from non-agile approaches (Benefield, 2008, Dybå and Dingsøyr, 2008).

There are some reports highlighting that agile methods are being used beyond their intended area of application which, according to these authors, represents a state of stability and maturity. But Boehm and Turner (2005), Ktata and Lévesque (2009b), Petersen and Wohlin (2009) and Cho, Huff and Olsen (2011) report that agile seems better suited for small stable projects. That said, VersionOne (2010) claims that even large organizations are beginning to adopt agile, while Forrester Research (West and Grant, 2010) found a negligible agile adoption rate difference between large and small organizations.

In summary then, the literature positions Scrum as a relevant, contemporary software development approach. It is the opinion of this thesis that Scrum is gathering growing interest from both practitioners and academics and this sanctions the necessity and relevance of academically biased theoretical contributions in this area.

2.4.7 Perspectives of Scrum implementation at the multi-project level

Earlier subsections point out that the literature seems to pay most attention to Scrum at the individual project level. Some suggest that more research is required to answer questions such as: "How can we optimize and how should we apply Scrum agile methods to large-scale and mission critical software development projects?" (Kim, 2007). Ambler (2009) acknowledges the usefulness of Scrum through its adherence to Lean principles but contends that current IT management frameworks are at odds with the requirements of agile practice, and stresses the need for more research into this area. The literature also claims a shortfall in the rigor of associated higher level management practices (Chen et al., 2009) when implementing Scrum. This is evident in literature such as Chan and Thong (2009), Jayawardena and Ekanayake (2010), and Cho et al. (2011) who discuss a wide array of areas of difficulty. Cho (2010) responds to these calls in literature and carried out a study similar in

nature to this thesis. The author concludes by presenting a theoretical model of management practices that are claimed necessary for the successful implementing and management of Scrum.

Dingsøyr et al. (2006) report on the successful implementation of Scrum across a two company development project. They find that Scrum afforded increased management overview, flexibility, and team motivation but had difficulty in other management areas such as cost estimation across the two projects. Some authors contend that few attempts are made to understand how agile development approaches are being used as product management and portfolio management vehicles; examples include Cervone (2010) and Young and Conboy (2013). It is recognised that the need for more clarity in this area persists (Dingsøyr et al., 2006, Hoda et al., 2008). Recent attempts to address these deficiencies have spawned professional bodies such as the Scaled Agile Academy (ScaledAgileAcademy, 2013) that aims to facilitate training for agile adoption at both project and program management levels (Leffingwell, 2013). This further supports claims of the existence of uncertainty of requirements of higher order management.

There are varying reasons for the calls for more research focusing on higher levels of management. From the multitude of lenses uncovered during the literature survey, the themes of uncertainty of practice and a required dramatic change in management culture, are explored further in the following subsections as these themes are considered more pertinent to this thesis.

2.4.7.1 Uncertainty of management practices

Nuer et al. (2005) report on success factors for changing from a traditional to a more agile orientated management and conclude that "variations between traditional and agile methodologies suggest that organizations must rethink their goals and reconfigure their human, managerial, and technology components in order to successfully adopt agile methodologies" (Nerur et al., 2005). Yahoo! software development company exhibits mixed reactions to agile management practice among the different development teams, according to Barton (2012). He concludes that a likely cause is that companies have not yet fine-tuned their higher level project management practices that can support agile. Hoda et al. (2010) and Yang et al. (2009) stress that an increasing number of organizations are experimenting with agile development, and assert that the leadership implications associated with such adoption are fertile ground for research but have yet to be more fully understood.

The literature indicates a scarce amount of conceptualizations or higher theoretical contributions focusing on higher level management practice (Dyba, 2005, Lee and Xia, 2010, Dingsøyra et al., 2012). Abrahamsson, Warsta, Siponen and Ronkainen (2003) maintain that effective ways to improve alignment between individual project level and higher management is generally missing from the literature. Jayawardena and Ekanayake (2010) contend that lack of expertise is one possible reason for slow adoption. Other authors suggest that "few organizations are psychologically or technically able to take on an agile approach rapidly and effectively" (Qumer and Henderson-Sellers, 2008), and that companies are not yet set up for agile and many do not yet know how best to integrate agile within their current organizational management frameworks (Pham and Pham, 2012).

Although their study was restricted by a small sample size of just 30, Ceschi, Sillitti, Succi and De Panfilis (2005) conclude that agile adoption is low due to a strong sense of uncertainty by higher level management and a general lack of well-established and widely accepted practice guidelines. A rather strong supporting claim is that management will not easily surrender control and perception of security afforded to them by plan-driven methodologies (Cohn and Ford, 2003). The authors go on to highlight the anxiety expressed by higher management for certain areas of management beyond the individual project level. Questions asked include: "How can we promise new features to customers? How can we track progress? How will the agile process impact other groups? And, When does the project end?" (Cohn and Ford, 2003). Vähäniitty and Rautiainen (2008a) opine a rather fledgling state of understanding of agile multi-team and multi-project environments.

These sentiments portray a tentative perspective of aligning agile with higher abstractions of management such as enterprise management and program management. Stepping away from the pragmatic concerns, the next section examines more conceptually focused literature.

2.4.7.2 A change in management perceptions

Coram and Bohner (2005) conclude that successful management of agile projects requires a major cultural shift for executive managers. They go on to highlight a current transformational challenge for middle managers, from line manager to advocates who can convince executive management of agile success. Leybourne (2009) is supportive of this point of view, and states that "this involves a dismantling of some elements of the traditional project management model in favor of experimentation and a shift in attitude by project managers away from the prescriptive, plan-based routine embedded in the documented

BoKs" (Leybourne, 2009). Early research by Lindvall *et al.* (2002), and more recently by Misra, Kumar and Kumar (2009) and Moe, Dingsoyr and Dyba (2010), opine that organizations require a period of time for transition from traditional hierarchical management to one more akin to support agile. Sutherland (2005) stresses that agile management should be implemented at a company level and not only considered for the individual project level. However, Chan and Thong (2009) and Cottmeyer (2009) find low company-wide adoption rates and posit that the most likely cause is the long standing management traditions and the organization's unwillingness to migrate, which in turn results in a "mismatch between project methodology and leadership style of the project manager" (Yang et al., 2009).

Augustine et al. (2005) and Highsmith (2004) set out to advance an area they refer to as Agile Project Management (APM) via a collection of management guidelines for the agile practitioner. Both pay careful attention to advancing the cause of changing roles and responsibilities of agile managers. For example, the analogy of sheep-dog (or shepherd) in place of a drill sergeant to represent the shift in mentality of an agile manager (Highsmith, 2004) is given, and the manager's additional responsibility set - which includes such functions as striving to let calm prevail within project teams (Cockburn, 2007), is suggested. It must be stressed that proponents of APM do not completely denounce traditional management. Instead they claim to be extending project management practices to better align with the agile software development environment.

The above subsections aimed to expose a gap in literature and further strengthen the need for research seeking answers to "How can we optimize and how should we apply Scrum agile methods to large-scale and mission critical software development projects?" (Kim, 2007). In other words, literature seems ill-equipped in providing rigorous practice guidelines for higher order management. These subsections are not masqueraded as a concise analysis of agile management literature, but are instead considered aligned with and supportive of the theoretical contributions envisaged for this study, which includes creating a perspective of multi-project program management practices within the context of agile software development. Having presented some key challenges from a multi-project management perspective, the next section sets out to describe the third major component of this thesis, namely program management.

2.5 A view of program management

This section reviews literature on program management which forms one of three central tenets of this thesis. It begins with a definition of program management and then provides a comparison between project, program and portfolio management. Thereafter, perspectives on popular project management frameworks and a recent call for rethinking their nature are reviewed. In light of these, this section then explores knowledge areas and practices, and life cycle models, that govern current project management. Recent developments in program management are exhibited and the section concludes with a view of program management within the context of software development.

2.5.1 A definition of program management

Program management is a discipline geared towards organizing and coordinating multi-project work, while a program can be defined as a collection of interrelated projects (PMI, 2008). Inspired by the need to maximise usage of limited resources, program management redefined the singular project focus (Lycett et al., 2004) to one of a multi-project mode of operation (Maylor et al., 2006, Pellegrinelli et al., 2007). Program management typically embodies a set of practices, guidelines, and principles that refers to organization and management, such that optimal coordination, control, and resource sharing is achieved between simultaneous projects (Aubry et al., 2007, OGC, 2007, Kerzner, 2013, Schwalbe, 2013).

The literature reveals that program management is sometimes perceived as a dossier of projects which are organized and directed such that the greater strategies for the company may be realized (OGC, 2007, Thiry, 2010). In this view, program management embodies practices that represent a subset of corporate governance activities (Blomquist and Müller, 2006) instead of simply being a means of organizing the project space. This notion becomes relevant when grouping related projects to form a program that collectively aims to serve common company objectives (Pellegrinelli, 2011, Unger et al., 2012).

It is important to note that this thesis investigates non-related software projects, each destined for a different client but sharing resources. Hence, in the context of this thesis, program management is perceived as an approach aimed at effectively orchestrating multiple, concurrent projects that are competing for resources. The next section compares the management focal areas of project, program, and portfolio management.

2.5.2 A comparison of project, program, and portfolio management

Often in literature the terms *program management* and *project management* are used interchangeably which, some opine, may be a consequence of the field being largely unsettled (Aubry et al., 2007, Artto et al., 2009). In the absence of a well-accepted definition, for the purposes of this thesis, a project has a predefined start and end date and objectives which should result in a desired end product. Each project is usually unique in that the outcome is a product or result not produced before (Ramasesh and Browning, 2014). Project management, for the purposes of this thesis, is a collection of practices that collectively endeavour to ensure that objectives are met within schedule and budget (Kerzner, 2013).

Program management generally seeks to provide a framework that "integrates and reconciles competing demands for resources, providing a context and control framework for projects of the program" (OGC, 2007), and bringing about *sense making* in an otherwise complex, multi-project mode of operation (Maylor et al., 2006). It does not have typical start and end dates normally associated with projects; instead it constitutes an on-going practice (Thiry and Deguire, 2007). Multi-project management is sometimes used as a synonym for program management (Thiry, 2004).

Program and project management are typically regarded as complementary (OGC, 2007) in that related projects have objectives that contribute towards the greater, singular outcome of the program. In this perspective, some prefer the term enterprise program management (Thiry, 2002, Pellegrinelli et al., 2007), while others use the term organizational project management and claim that "the goal of organisational project management is not just to deliver projects on time, on budget and in conformity with technical and quality specifications. The goal is to create value for the business" (Aubry et al., 2007).

Portfolio management practices are often perched at a higher organizational level. Portfolio management (sometimes also called project portfolio management) refers to the logical grouping of projects and programs under a common leadership structure (Brown, 2008) in order to enable their management as a portfolio of business investments (Vähäniitty and Rautiainen, 2008b). "Portfolio managers help their organizations make wise investment decisions by helping to select and analyze projects from a strategic perspective" according to Schwalbe (2013). Hence, portfolio management is sometimes likened to a negotiation-driven decision-making process (Thiry, 2010). The alignment of project, program, and portfolio goals with the strategic objectives of the company is encouraged (Aubry et al., 2007, PMI,

2013). With portfolio management the primary concern lies in aligning interrelated projects with company strategy, and typically entails the selection of programs and projects that best satisfy the long-term goals and strategy of the company (Blomquist and Müller, 2006, Ramasesh and Browning, 2014). A simple differentiation is that program management asks: Are we doing the projects right? whilst portfolio management asks: Are we doing the right projects?

The diagram below illustrates the comparison in organizational level with some major practices and is included for readability. These practices were synthesized during investigation of program management knowledge areas and practices and these will be elaborated in much more detail later in section 2.5.6.

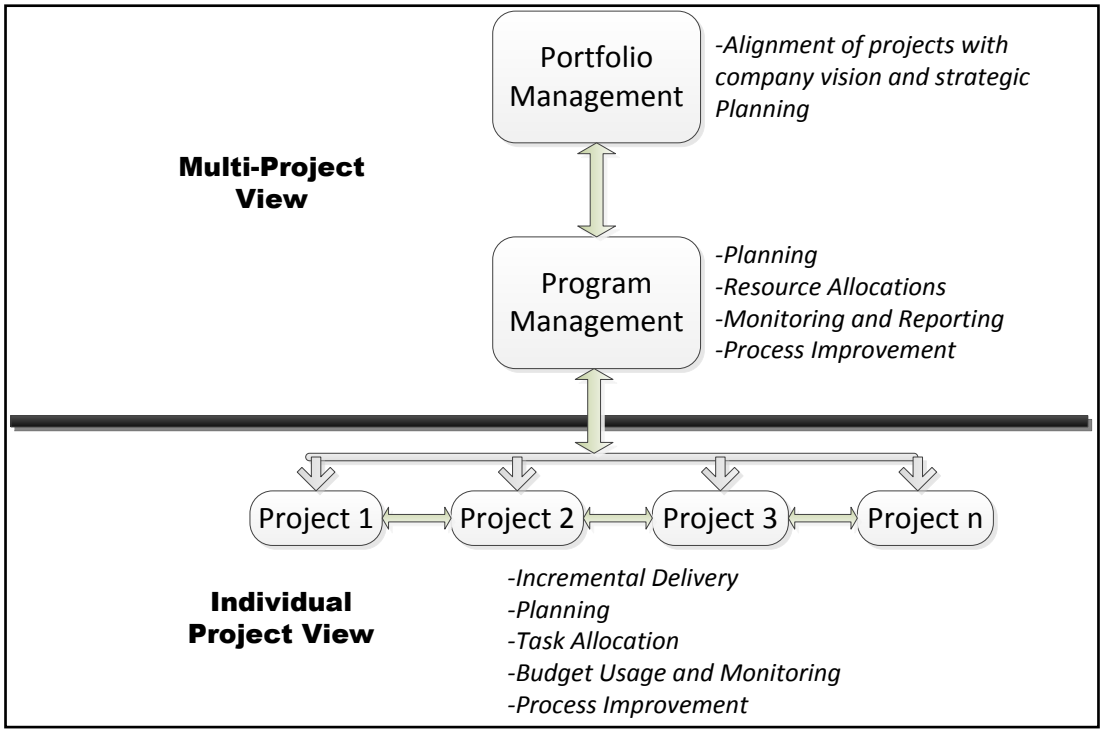


Figure 5: Portfolio, program and project management layers with example practices. Source: Researcher's own construction

The next subsection examines literature to establish a perspective of the current management frameworks that underpin the practices of program management.

2.5.3 Overview of management frameworks underpinning program management

A project management framework offers a prescriptive lens though which project-based work can be controlled (Ahlemann et al., 2013). Such a framework often adopts the structure of a staged approach, with guidelines for best practice suggested for each stage (Meredith and

Mantel, 2011, Kerzner, 2013). Stages often embody a blend of sequential and iterative sets of activities to achieve a predetermined objective. Sometimes project management frameworks are underpinned by principles and themes which impact on the nature of activities.

A contemporary popular project management framework is the Project Management Body of Knowledge (PMBoK) (Schwalbe, 2013, Hinojo, 2014, Usman et al., 2014), with the Project Management Institute (PMI), an international professional body, serving as custodians of the framework. PMBoK can be viewed as a set of prescriptive guidelines and is often claimed to be a prevalent international standard for project management (Matos and Lopes, 2013, PMI, 2014). PMBoK Guide version five is the latest incarnation and aims at providing standard terminology and guidelines for project management (Levin and Green, 2010, PMI, 2014). PMBoK derives five stages or process areas and thirteen knowledge areas for use as a project management lens. The PMBoK Guide is commonly perceived as a framework of best practice guidelines aimed to improve project management (Reich et al., 2008, Whitaker, 2010, Kerzner, 2013, Schwalbe, 2013).

A recent development is the Software Extension to PMBoK Guide (PMI, 2014). The purpose of this extension is emphasized by PMI as a way to achieve greater alignment with modern software development approaches such as agile (PMI, 2014). One can infer from this proclamation that existing PMI-based frameworks were not entirely well suited for agile. Literature is sparse with regards to opinions on its adoption and effectiveness.

An alternative framework to PMBoK is Projects in Controlled Environments (Prince2) (Barker, 2013, Schwalbe, 2013). Prince2 is often described as the de-facto standard for project management in the UK (APMG-International, 2012). Prince2 is a process-driven project management methodology. It views the project as a set of process areas, each consisting of guidelines for progressing through the stages of a project. This is a prescriptive methodology with a clearly defined framework for directing a project (Grant et al., 2010). Some of the perceived benefits of Prince2 are greater control of project related resources, a defined framework of practice, common lexicons, and improved risk mitigation due to more stringent management control mechanisms (Grant et al., 2010).

While PMBoK and Prince2 apply to project management in general and provide generic knowledge areas of practice, there are other frameworks that are more aligned to software project management. One such framework is the Capability Maturity Model Integration (CMMi). CMMi is maintained by the Software Engineering Institute (SEI) and represents one

of the more popular software process improvement approaches (Glazer et al., 2008, Sommerville, 2011). CMMi is often referred to as a process improvement framework for software products and services, and consists primarily of best practice and guidelines for each phase of the software project life cycle (Marcal et al., 2007, SEI, 2013). Highsmith (2002b) claims that approaches like CMMi offer the perception of a well-defined process in which tasks are clearly defined, outputs accurately measured, and processes refined until they are repeatable. It should be noted though that CMMi is reported as being more popular among larger companies that typically exhibit a larger arsenal of resources (Laporte et al., 2008a, Lee and Yong, 2013).

2.5.4 Perspectives on contemporary management frameworks

The literature shows that program management is often referred to as a performance focused paradigm whereby the objectives of schedules, economic and control mechanisms dominate (Thiry, 2002, Ahlemann, 2009). This is usually attributed to the legacy of early professional management bodies such as PMI and OGC and their project management frameworks. These project management frameworks have recently come under intense scrutiny. Authors such as Denning (2009, Denning, 2010a, Denning, 2010b), Hodgson (2004), Hodgson and Cicmil (2008), Martin and Austen (1999), Hoebeke (2000), and Martin (2005, Martin, 2011) make a desperate plea for investigations into revising existing management philosophies, stating that these are required if companies are to reverse the current trend of poor project success. The literature sometimes portrays these project management frameworks as rationalist (Ahlemann et al., 2013) or industrial process type strategies (Espinosa et al., 2009), and accusations are made that they lack a degree of leeway to nurture innovation and creativity in the project management practices (Moe et al., 2010, Sewchurran et al., 2012).

Winter, Smith, Morris and Cicmil (2006), Fernandez and Fernandez (2008), Sauer and Reich (2009), and Crawford, Morris, Thomas and Winter (2006) suggest that both research and practice should move beyond the singular aphorism of mechanistic application of defined life cycles. During the infancy of software engineering, a famous researcher once stated that “the dismantling of rigid prescriptive planning-based project management models, and the acceptance of more adaptive modes of managing projects, is becoming more accepted” (Royce, 1970). Reich et al. (2008) lament that current advice on how best to improve software project success has largely remained unchanged in focus even though there has been no plausible impact on persistent poor project performances. Some literature highlights that

the current thinking that underpins most project management often results in symptoms of "[…] lack of acceptance in practice, limited effectiveness, and unclear application scenarios" (Ahlemann et al., 2013).

Referring to deficiencies of current project performance management, Toor and Ogunlana (2010) find that traditional measures of the iron triangle (on-time, under-budget, and according to specification) are no longer core drivers for project success. Although these objectives are important for project orientated work environments, other considerations are emerging. For instance, Sauer and Reich (2009), Nerur and Balijepally (2007), and Babb and Nørbjerg (2010) support the inclusion of project perspectives and subsequent practices that embed knowledge transfer and learning processes. The next subsection presents literature that extends this debate.

2.5.5 The call for refocusing project management frameworks

The *iron triangle* criteria of cost, time, and quality when applied to IT projects was scrutinized some time ago by Atkinson (1999) who proposes supplementary criteria to establish project success, namely stakeholder benefits, organizational benefits, and information systems quality metrics (such as reliability and quality in use). This is an early example of literature that signalled a growing belief in the necessity of incorporating other concepts predominately defined by a stronger behavioural and improvised focus. This form of management instead nurtures traits such as "creativity, intuition, adaption, compression, innovation and learning" (Leybourne, 2009). This perspective is mirrored in the following quote:

> "The concepts of transfer prices, of profit and loss centres are becoming increasingly less relevant…[Instead, focus should turn to] value adding work systems for customers in terms of throughput time, intrinsic quality, value and the effort the customer is prepared to spend for product and services" (Hoebeke, 2000).

More recently, Denning (2010a) suggests that prescriptive management frameworks, and their subsequent practices, on their own are not entirely favourable to the modern organizational environment. As a possible reprieve to these allegations, a set of guiding principles to supplement these management frameworks is suggested. These principles are proposed in the form of recommendations and are juxtaposed with more traditional management philosophies in Table 4:

Table 4: Comparison of traditional management thinking and recommended change. Source: Adapted from Denning (2010a).

Traditional management thinking	Required change in management thinking
Maximise value for the shareholder	Strive to provide the client with a "pleasant" business interaction experience
Managers should control employers	Allow more self-management and self-organizing among teams
Work is handed down hierarchically	Move towards flexible projects whereby value is created for the client in short deliverable cycles
Efficiency is core focus	Focus on providing value for client
Communication occurs in a top-down hierarchical direction	Encourage interactive communication and problem solving
There are strict communication channels	Allow for transparency of project needs and status
Targets are set and handed down	Create a work environment conducive to continuous improvement

Viewed against the backdrop of software projects, Cockburn (2007) makes the strong statement that mechanistic approaches to software development project management are usually flawed due to people's usual inability to exactly replicate and repeat tasks. Software projects rarely have exactly the same parameters and often require a fair dosage of creativity and intuition, among other qualities. Hence, publications like Hoda et al. (2013) and Babb and Nørbjerg (2010) see the need to supplement traditional, prescriptive management approaches that underpin current software projects. "Battlefield commanders succeed by defeating the enemy (the mission), not conforming to a plan" according to (Highsmith, 2002b). Software development is largely a mental activity and software emerges from the mind of the creator (Highsmith, 2000a), although the encompassing environment should be conducive to sustaining these activities (Cockburn, 2002). Supporting this viewpoint, Nerur et al. (2007), Fernandez and Fernandez (2008), Leybourne (2009), and Yang et al. (2009) feel that the more traditional, mechanistic management style is becoming more restrictive for software projects, while Babb and Nørbjerg (2010) view classical software development approaches as a tenet of technical rationality. Technical rationality is a concept derived from older management doctrines which view individuals as a source of constant error and unpredictable behaviour; hence the need for strict adherence to methodology to control the effects. These and other authors feel that this is a limiting scientific theory which underpins many of the well-entrenched software project management approaches.

This raises the following question: In reaction to these called upon practices, what might contemporary program management resemble? A perspective is tendered next.

2.5.6 Key knowledge areas and practices for program management

This subsection provides a synthesis of program management knowledge areas and practices uncovered during the review of literature. This serves as an early lens for the investigative stage by informing the researcher of terminology and concepts currently considered in program management.

Hanford (2004), Lycett et al. (2004), Blomquist and Müller (2006), Brown (2008), Jonas (2010), and Thiry (2010) highlight key knowledge areas and practices for program management. These are synthesized in Table 5 below:

Table 5: Key generic knowledge areas and practices in program management. Source: Researcher's own construction

Knowledge Area	Practices
Organization	Establish roles, hierarchy, and responsibilities for individuals or teams
Management	Establish objectives, coordinate activities, and establish reporting mechanisms
Financial management	Estimate and report on expenditure across projects
Risk management	Take actions in lieu of the absence of clarity and/or changing project characteristics such as tools and people
Resource allocation	Do multi-project resource planning, Human Resources Management, and resource reallocation in reaction to short-term change requests
Infrastructure	Define tools and methodologies used across projects
Planning	Review of individual project plans resulting in a holistic image of tasks, schedule, and deliverables
Communication	Design mechanisms for knowledge transfer between individual projects, project to program, and projects to business

The above table is not designated an exhaustive list. Nonetheless, it does provide clues as to the type of work typical of program management, and it helps the reader to acclimatise to the data analysis sections provided later in this thesis. It is envisaged that program management at the case sites may resemble some of these activities, although undoubtedly new ones will emerge.

2.5.7 Suitability of procedural program management life cycle models

The practices alluded to above are generally operationalized within companies via a program management life-cycle. The literature sometimes defines the life-cycle as a stepwise, procedural process (Lycett et al., 2004, Thiry, 2004, Brown, 2008, Ahlemann, 2009, Levin

and Green, 2010) consisting of the generic stages of initiation, planning, delivery (including cycles of monitoring and controlling), and closure.

The diagram below provides a temporal illustration of the stages of a typical project life-cycle. For the sake of improved readability, a handful of activities are shown for each stage.

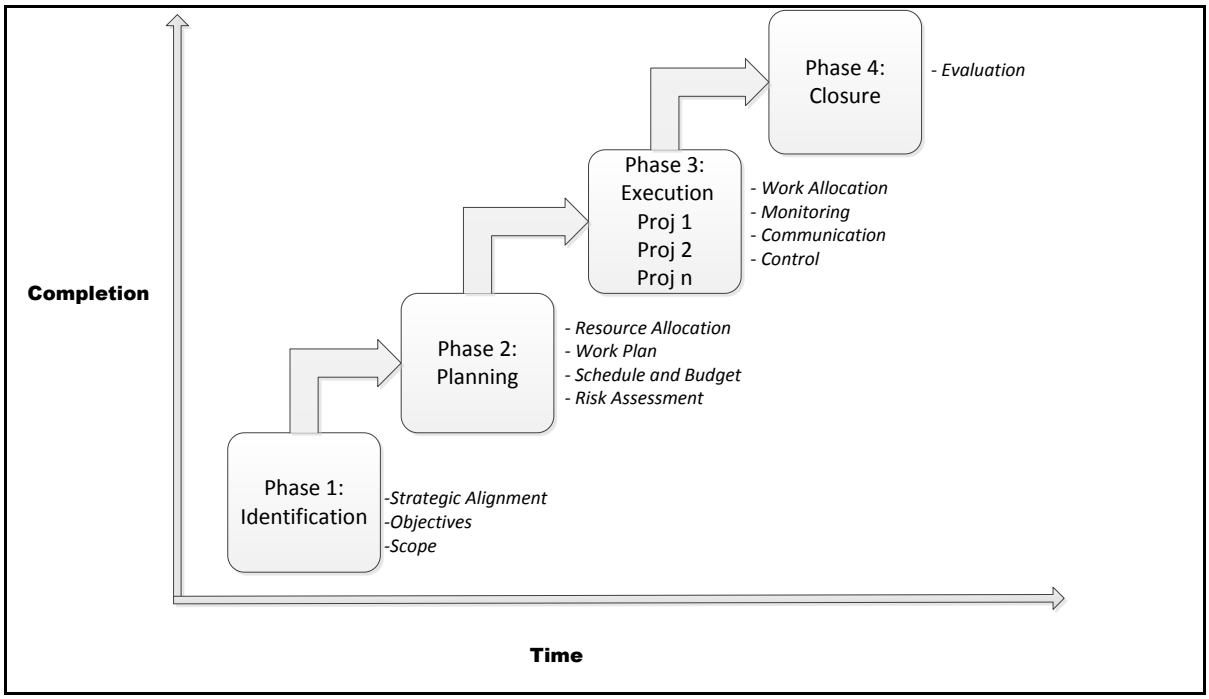


Figure 6: Program management life cycle phases.
Source: Researcher's own construction

It has been suggested that this procedural life cycle model is most likely a remnant of management practices arising from the campaigns of standards organizations such as PMI and OGC (Pellegrinelli et al., 2007, Levin and Green, 2010). In this point of view, program management is viewed as an extension of project management practice, that is, as a large project which is divided into multiple, smaller projects; hence the justification of the direct application of project management life cycles in the program management context. (Pellegrinelli et al., 2007). While the current program management discourse embodies static or once off practices such as initial planning, at the same time it needs to exhibit characteristics which render it an emergent and adaptable process in response to a changing project environment. A framework of practice, instead of linear sequence of tasks, may be more apt as a vehicle for describing program management. Furthermore, any representation of program management should resemble a network unifying such constituents as people, artifacts, methodology, and a collection of lessons learnt from experience. Even though program management might not necessarily resemble a strict procedure, it can still be

considered as a network of activity. For this reason, this thesis does not whole heartedly subscribe to a strict rationalist perspective of program management with staged, sequential flow of practices.

2.5.8 Recent developments in program management

The Office of Government Commerce (OGC, 2007) based in the UK, Project Management Institute (PMI, 2013) based in the USA, and Project and Program Management (P2M, 2008) based in Japan, are examples of international professional bodies that provide practice guidelines for project and, by extension, program management. Levin and Green (2010) contest that program management is making in-roads towards becoming a field of greater maturity and standardisation. Some argue that the aforementioned are the most pragmatically subscribed to practice guidelines (Thiry, 2010) and often feature as lenses for academically inclined studies; for examples see Pellegrinelli et al. (2007) and Artto et al. (2009). The literature depicts areas of discourse that include, but are not limited to, better understanding of the complex relationships between project, program, and portfolio management and the organization (Aubry et al., 2007) in order to gauge program management performance (Müller et al., 2008, Jonas, 2010) and to elucidate and generalize practices and roles (Unger et al., 2012).

Lycett et al. (2004) conducted an extensive review of program management practices in UK-based companies. Their findings reveal problematic areas stemming primarily from excessive project control, insufficient flexibility to cater for changing company strategy, and problematic interfaces between projects. Lycett et al. (2004), Pellegrinelli et al. (2007), and Pellegrinelli (2011) make the assertion that the major difficulties experienced in achieving effective program management practices stem from flawed assumptions; for instance, that program management practices can be uniformly applied across environments, and program management is scaled-up project management. A relatively strong claim on the apparent deficiency in program management is made that "after more than 35 years of working for organizations of all sizes and more than 15 years of management consultancy […] I have come to realize that many organizations still don't understand how to integrate their business practices" (Thiry, 2010). Supporting this claim, Jonas (2010) states that despite its long lineage, a well-accepted description of program management functions and strategic objectives has yet to be seen.

The literature highlights another perspective, namely that program management aims to bring about change to organizational strategy (Thiry, 2002, Lycett et al., 2004, Thiry, 2004). These authors contend that program management should not focus entirely on technical practices (such as schedule and resource allocation), but also on long-term management, and it should serve as a catalyst for change within the company. For them, program management lies within the context of strategic management. Some propose a model combining concepts from knowledge management and organizational learning with performance measurement to expedite this link (Thiry, 2002, Thiry, 2010). The OGC (OGC, 2007) posits program management as the primary structure to reduce the tension between company strategy and daily operations management or project level work (Pellegrinelli et al., 2007, Aubry et al., 2009). In this perspective program management is seen as the arbitrator for change within the company.

Some authors reject the rationalist, positivist outlook of project and program management (Aubry et al., 2007, Jonas, 2010, Pellegrinelli, 2011). Instead, these authors subscribe to the notion of trying to understand management through theoretical lenses derived from other disciplines such as sociology and organizational theory. This notion is elaborated in the assertion that there is a need for "[…] a distinct programme management model, grounded in a view of social reality as continually constructed through the actions and interactions of individuals […]" (Pellegrinelli, 2011), and in the statement by Aubry, Hobbs and Thuillier (2007) that:

> "We argue that the study of such complex relationships within an organisation should turn away from the traditional positivist approach to a new conceptual framework. The proposed theoretical framework [of this study] draws from three complementary fields – innovation, sociology and organisational theory – to form an innovative understanding of the PMO and organisational project management" (Aubry et al., 2007)

Pellegrinelli et al. (2007) and Artto et al. (2009) conducted a review of existing literature related to project and program management. They found that program management draws from various theoretical bases which they believe evidences an active field of enquiry, whilst project management tends to draw mainly from the area of product development. Pellegrinelli et al. (2007) contend that despite almost a decade of practitioner and academic work, program management theory and practice still remains largely shrouded with uncertainty. A similar point of view is mirrored in Blomquist and Müller (2006) who claim that program management structures implemented in companies are mostly unclear.

The subsection above aimed to orientate the reader to the broader discourse of program management and, via contemporary literature, showcase an understanding of the state of the field. The next subsection examines literature on the use of program management as a lens on agile practices.

2.5.9 Recent developments in program management in software contexts

Studies such as Vähäniitty and Rautiainen (2008b) attempt to elucidate portfolio management within Scrum project contexts in order to demonstrate an impetus towards theorizing Scrum outside the project level. Hanford (2004) stresses that traditional management approaches are failing in balancing delivery of software systems, changing business models, and shifting organizational structure. The author suggests that the underutilized program management body of knowledge may prove useful in this problem area. Parolia et al. (2011) found that program management is used effectively in larger, Indian-based software companies for resource sharing among projects; and the associated business objectives and overall operations of the companies also improved as a result of its use. Apart from a few sources such as these, the literature is fairly silent on using program management as a lens for agile software development management practices.

2.6 Summary

This chapter presented the findings of a survey of the literature in three major areas of research, namely, software SMME management characteristics, Scrum software development, and program management. Each of these areas was dealt with separately. The first major section focused on defining SMMEs (section 2.3.1) and evaluating their potential contribution to the economy (section 2.3.2). Despite the researcher's best efforts, it was found that there is a lack of academic and statistical literature on South African software SMMEs (section 2.3.3). To accommodate for this deficiency, publications from the USA and the EU were sourced. This literature indicates that SMMEs face challenges that stem primarily from limitations in manpower and resources and this results in a particular shape of management practices. These management practices were synthesized (section 2.3.4), not concisely but to an extent deemed sufficient enough to provide a preliminary mould for the thesis fieldwork. Apart from the handful of publications that were cited, there was a shortage of publications focusing on management practices of software SMMEs, and no studies were found that focus specifically on how these particular SMME management practices impact on the software development process and on program management.

The chapter then progressed to the second primary research area, Scrum software development practices. This section began by providing a definition of agile (section 2.4.1) and of Scrum (section 2.4.2). A conceptual link between agile and Lean principles was then established (section 2.4.3). This allowed an improved understanding of principles underlying agile and, by extension, enabled improved sense-making of the nature of the practices of Scrum (section 2.4.4). The Scrum roles, practices, artifacts and metrics were then synthesized (section 2.4.5). A proposition of what might be witnessed during the fieldwork was presented in order to prepare the researcher and the reader for better interpretation of the data in later chapters.

The literature reveals that agile is considered an important recent development in the software industry, and Scrum is a relevant contemporary software development approach stemming from agile (section 2.4.6). It was noted that Scrum often features in exploratory type research and large corporates seem to also be experimenting with agile. In unveiling this perception, it was argued that there is a the need for deep exploration type research endeavours.

The application of agile at higher levels of management and across multiple projects are two areas that emerged as being in need of more research contributions (section 2.4.7). The literature presents a multitude of lenses through which to view this problem, but only two of these lenses were considered relevant to his thesis, and these were then discussed further. The first lens took on the problem of uncertainty of management practices (section 2.4.7.1). Here the literature points to a degree of deficiency in existing management frameworks that seek to accommodate agile. In addition to this, there also appears to be a disarray between organizational and project level management, and there are calls for a change in management thinking if agile is to be successfully adopted (section 2.4.7.2). It was noted that these two lenses represent areas lacking in well accepted and well defined conceptualizations of multi-project practices derived from sound academic research; and both of these subsections were seen to serve as forerunners to the more detailed insights in the program management section that followed.

The chapter then progressed to the third and final primary research area of program management. After providing a definition of program management as being a multi-project mode of operation that aims to oversee concurrent, competing projects (section 2.5.1), the following subsection (section 2.5.2) provided a comparison of project, program, and portfolio management as hierarchical layers of management. This afforded the reader with an indication of the focal area of program management for this thesis. Thereafter a point of view

pertaining to contemporary project management frameworks was unearthed (section 2.5.4). Based on this point of view, the call for refocusing traditional management thinking to cater for more than *iron triangle* objectives was presented (section 2.5.5). A view of program management knowledge areas and practices was then presented (section 2.5.6). It was argued that whilst the literature portrays program management as defined life cycle models (section 2.5.7), this thesis subscribes to the idea that program management has a more emergent nature that differs from the opposing idea of a pure rational, prescriptive nature. This thesis aligns with the notion of program management practices being more dynamic and that they are shaped in reaction to pressures from project parameters but at the same time conform to SMME and Scrum characteristics. However, literature is just gaining traction in understanding the dynamic nature of these practices. Current perspectives of program management's philosophical underpinnings and recent research lenses, popular areas of research, persistent challenges and the nature of program management were also presented (section 2.5.8) in order to provide the reader with a high level view of the program management discourse and how it is positioned within this thesis. Finally it was noted that there is a lack of literature focusing on program management in the context of software projects (section 2.5.9).

The literature review sought answers to questions what are the characteristics of software SMMEs?, what are the key practices in Scrum? and, what are the key knowledge areas for program management in software development environments? It then endeavoured to explore any existing conceptual interface between the three core areas. In conclusion, this literature review discovered that the nature of program management in the context of software SMMEs practicing Scrum, is largely unknown. Despite the researcher's best efforts, no studies where found attempting to conceptualize and present the interface between software SMMEs, Scrum and program management. Hence, it is argued that there is ample room for more research aimed at understanding how Scrum software development can be linked to program management in the context of software SMMEs.

CHAPTER 3: Unpacking the research framework

3.1 Introduction

The research framework is a theoretical image of the object of study. It emphasises the logic, and the logistics, of the research undertaking and essentially represents a plan for progressing from the research questions to the results. Palvia et al. (2006) feel that ideally, a research framework should be designed based on existing theoretical lenses. However, they also contend that there is a deficiency of generally well accepted theory in the field of IT; this in turn encourages IT researchers to employ a research framework to help structure key components of the research undertaking and to facilitate theory formulation.

In line with this claim, a framework to structure and highlight the relationship between the core research components that is derived from Denzin and Lincoln (2008) and Creswell (2009), is employed in this thesis. The key components of this framework are as follows:

a. A philosophical underpinning or assumption about the nature of reality (ontology) and the researchers understanding of the world (epistemology)
b. A strategy used to conduct the study, either qualitative or quantitative
c. Methodologies that operationalize the framework
d. Techniques for data collection, data analysis, interpretation or inference of theory, validation of framework, and write-up

This chapter describes the research framework. It begins with a description of the approach used for research question formulation and highlights the main and subsidiary research questions that emerged out of the literature review phase. Thereafter, the major components of the research framework, namely the philosophical underpinning, research strategy, methodology, and techniques for data collection, are described. The subsections that follow offer more detailed descriptions of each of these four research framework components. First, the philosophical underpinnings elected for this study are explored, followed by the research strategy, operationalizing methodology. Finally data collection techniques are discussed. Data analysis is left for later chapters: In Chapter Four, the use of the unusual lens of Activity Theory is elaborated; and in Chapter Seven, the use of Concept Analysis as a data analysis tool is discussed. Table 6 depicts the major components of the research framework as discussed in this chapter, with the exception of data analysis as mentioned:

Table 6: Summary of the nature of components of the research framework

Component	Nature	Chapter section dealing with this component
Research question forumulation	Follows Denzin (2002)	3.2
Ontology	Interpretivism	3.3
Research strategy	Qualitative	3.4
Epistemology	Relationship concepts of Activity Theory	Covered in Chapter Four
Methodology (Research design)	Descriptive case study at two case sites	3.5
Data Collection	Ethnographic techniques: o observation o limited participation o interview (semi-structured and open-ended) o project artifact analysis	3.6
Data Analysis	• Activity Theory analysis of the two case studies	Activity Theory is covered in Chapter Four
Data Interpretation	• Thematic analysis • **Concept Analysis is used to further refine the thematic analysis**	Thematic and Concept Analysis is discussed in Chapter Seven
Theory Formulation	• Guidelines from Gregor, Llewelyn, and Whetten • Inference: o Abductive reasoning (for early propositions) o Inductive reasoning (for data analysis)	3.7

Some of the sources cited in this chapter are drawn from other fields of science, such as social science and operations research. This digression allows for a richer mix of more suitable research methodologies when compared to strictly IT sources, and plays a pivotal role in honing the research framework of this thesis.

3.2 Formulation of the research questions

The formulation of research questions in this thesis was not coincidental and there was a fair degree of mechanics implicated in the process. This thesis implements a procedure for research question formulation that is based on the work of Denzin (2002), as follows:

1. Identify the real-world problem
2. Examine why this problem is becoming an issue or challenge
3. Locate the environments where persons experience these challenges
4. Find answers to how these experiences occur, not why they occur
5. Formulate the research question

For this thesis, steps one and two were achieved through a review of relevant literature. Step three involved contacting and requesting software SMMEs to volunteer as case studies. A point of departure from Denzin's procedure is that thesis does not strictly conform to problem identification and resolution in the customary research approach. Instead it contends that there is a lack of theoretical contributions in the contextual area of this study and seeks to make a contribution therein. Hence, step four of this study was more akin to finding answers to both *how they occur* and *why they occur*. Research question formulation, as indicated in step five, was done after steps one and two; but the research questions were subjected to further refinement during the course of the fieldwork - which Forsythe (1999) and Creswell (2013) feel is a natural occurrence in qualitative studies.

Utilizing the procedure above, the research questions that emerged as the product of the literature review, and that were influenced during fieldwork before concretization, are as follows:

MAIN RESEARCH QUESTION:

How is program management carried out in software SMMEs practicing Scrum?

SUBSIDIARY RESEARCH QUESTIONS:

1. *Why do the characteristics of management practices of software SMMEs influence program management?*
2. *How does Scrum influence program management?*

3.3 Ontology: Brief exploration of candidate philosophical underpinnings for this study

There are several available philosophical paradigms for underpinning research endeavors. Researchers such as Burrell and Morgan (1979), Giddens (1984), Jackson et al. (1991), and Deetz (1996), provide commentary on the developments and juxtaposition thereof. A research undertaking is more often than not grounded in one of three popular philosophical underpinnings; namely, positivism, critical research, or interpretivism (Deetz, 1996, Mingers and Broklesby, 1997, Klein and Myers, 1999, Merriam, 2002, Jackson, 2006, Stahl, 2013). A cursory discussion of each is now provided as a mechanism for leading up to the choice made for this research undertaking.

3.3.1 Positivist research

A research undertaking may be deemed positivist if it exhibits formal propositions, hypothesis testing, measurable variables, and established inferences from an elected sample population (Klein and Myers, 1999). Positivist research, by and large, is perceived as research that:

a. Works toward achieving universal or general laws within a field of science
b. Acknowledges an objective reality independent of the researcher
c. Expresses its theory's propositions statistically (consistent) and is considered an accurate representation of the real world (empirical)
d. Produces knowledge that is considered factual and not based on opinions, perceptions and beliefs.

(Lee, 1991, Lee and Baskerville, 2003, Carlsson, 2006, Smith, 2006).

Whetten (1989) maintains that positivist research generally requires some form of hypothesis testing. To illustrate this point, Mingers (2004) uses an example of a chemical experiment in which a precise environment needs to be created before a chemical reaction occurs. The outcomes of this experiment are then observed and compared to existing knowledge. Results are often generalizable. A successful experiment depends on the ability of the scientist to replicate the environment correctly. Some researchers feel that positive research is not well suited for research within the field of social or behavioral science (Lee, 1991, Mingers, 2008) since an artificial environment cannot easily be created, nor can the results be generalized outright (Mingers, 2004).

3.3.2 Interpretive research

Interpretive research is perceived by some as affording more in-depth investigations into a variety of contexts (Klein and Myers, 1999) and it is viewed as a popular variation for IT-related research (Klein and Myers, 1999, Lee and Baskerville, 2003, Walsham, 2006b). Some authors see interpretive research as a suitable conduit for providing new insights into phenomena emerging from management practices (Gregor, 2006), because it mainly seeks to understand the as-lived experience from those who influence and are influenced by the phenomena under investigation (Walsham, 1995, Myers, 2013). Unlike positivism, typically there are no predefined variables. Instead, interpretive research attempts to assign meaning to phenomena from people's understanding of the situation (Walsham, 2006a). Rowlands (2005) claims that interpretive researchers do not seek to uncover an objective representation

of reality but strive to understand the phenomena through a shared representation between the researcher and the interviewee. Interpretive research is affirmed as an approach whereby "knowledge of reality is gained through social constructs such as language, consciousness, shared meanings, documents, and other artifacts" (Klein and Myers, 1999). The epistemology of interpretive research defines knowledge as a social construction of actors (Walsham, 2006a) and the very name interpretivism stems from the concept that people's interpretation of reality shapes the meaning they attach to it (Jackson, 2006).

Interpretive studies do not generally set out to identify and showcase universal laws; instead, they aim to describe observed events shaped by the researcher's theoretical stance or lens (Walsham, 1995, Smith, 2006). With interpretive research, the applied notion is that people have their own subjective interpretation of their environment and the communication of their interpretation is the essence of the investigation (Lee and Baskerville, 2003, Walsham, 2006a). In comparison to positivism, interpretivism posits that "reality is not the fixed, single, agreed upon, or measurable phenomenon that is assumed to be in positivist, quantitative research" (Merriam, 2002). However, interpretive research cannot always highlight the underlying cause or reasons for people's as-lived experiences. For this, a critical research approach may be better.

3.3.3 Critical research

Some authors suggest that gravitation towards critical research tends to be a result of disillusionment with positivist and interpretivist paradigms (Oliver, 1992). This may be better explained by tracing the roots of critical research back to shortly after the Second World War. The founding members of the Institute of Social Research at the University of Frankfurt were primarily Jewish and most of them fled to America at the outset of World War Two. Having re-established themselves at the Institute in California, they saw many problems with the dominant American social research culture of *taken for granted* empirical practices (Kincheloe and McLaren, 2003). According to these authors, this standpoint led to the rise of critical research during the 1960s. It began to gather momentum due to individuals wanting change from being comfortable in a world of domination and subordination, to a world of equality and independence. Critical research normally allows questioning of how:

> "issues of power and justice and the ways that the economy, matters of race, class and gender, ideologies, discourses, education, religion and other social institutions and cultural dynamics interact to construct a social system" (Kincheloe and McLaren, 2003).

Critical researchers assume that people's perception of reality is influenced by political, economic, and social change (Sayer, 1992, Merriam, 2002, Kincheloe and McLaren, 2003, Creswell, 2009); and that critical research seeks to uncover the tension that exists in social and cultural groupings and progress towards deeper understanding of the causes of this tension (Sayer, 1992). More so, critical research strives towards unearthing *how* these tensions influence humans, and endeavors to uncover the forces that are preventing change (Kincheloe and McLaren, 2003). With regards to social and behavioral research, Oliver (1992) stresses that positivist and interpretivist based research is largely conducted by people who have power on people who do not. This differs from critical research which is sometimes considered emancipatory[1] since the social sample is encouraged to actively participate in the research to bring about change to their own current situation (Stahl, 2013).

3.3.4 Potential shortcomings of positivist and critical research

Bhaskar (1978, Bhaskar, 1998a) criticizes positivism using the argument of the epistemic fallacy, which he defines as the undesirable phenomenon that occurs when a study's epistemological assumptions are mistaken for (or reduced to) ontological assumptions. To clarify, in positivist research the researcher will typically conduct experiments to prove a hypothesis. The epistemology of positivism assumes that only what can be observed exists in reality and that what cannot be observed does not exist. Sometimes there are unobservable events that cause the situations or phenomena being experienced, but positivism is not usually geared to acknowledge the existence of unobservable events due to its basic premise that these cannot be easily observed (Archer, 1998). From this, one can deduce that the ontology of positivism is generally dictated by only that which can be observed. Some authors concur with this proposed phenomenon of epistemic fallacy, and warn of the potential dangers if the researcher fails to realize the differences between epistemology and ontology in their research (Dobson, 2002, Mingers, 2004, Wuisman, 2005, Carlsson, 2006, Smith, 2006). These authors suggest that, for the positivist researcher, observed phenomena are a true indication of reality; whilst for the pragmatist, knowledge is considered useful if it leads to results, without much regard to why reality exists in that manner. Another shortcoming with the positivist stance is the act of bringing about artificial conditions for the purpose of observations (Mingers and Broklesby, 1997, Mingers and Rosenhead, 2004). These observations may not always be accurate for social or behavioral science type research due to potentially changing people's

[1] The term emancipation is defined in this thesis as the empowerment of people to ask: "What do we want?"

perception, and due to scientists being unable to consistently and accurately replicate these open systems in other studies.

With regards to critical research, the focus of this thesis is not solely on the social structures and tensions that exist within a software development project. Although this thesis does not disregard the presence of social structures and tensions, they are not the central focus of the study. Furthermore, critical research mostly aims to enroll people and turn them into active participants who are seeking to bring about change within their social ecosystem. The people and practices under investigation in this thesis were not envisioned to be using this study as a vehicle to directly empower themselves to improve on their current situation. Instead a more exploratory nature was envisaged for this thesis.

3.3.5 Motivation for the use of interpretive research as philosophical underpinning

This thesis does not set out to prove strictly predefined propositions, nor does it aim to generalize results across all software SMMEs in a manner mostly subscribed to in positivist research. Unlike critical research, it does not aim to strictly focus on influences on social behavior, tensions, or politics of the software teams and management. Instead, this study sets out to explore and to conceptualize program management practices in software SMMEs. This requires the immersion of the researcher in the as-lived reality of software teams and managers working on real-life software projects, in order that their practices can be observed via a variety of techniques over an extended period. Consequently, interpretive research has been chosen to underpin this study.

The study will be done at at two case sites, over a prolonged period, to enable in-depth investigations within natural project settings, and so as to not compromise observations as would be more likely if the study was conducted in an artificial setting. Attempts will be made to preserve the participants' perspectives and ensure the researcher portrays *their* story. Furthermore, in a high innovation environment like software development, various artifacts and metrics had a strong influence on individual perceptions of tasks and performance. Interpretivism appreciates that each individual undoubtedly has their own point of view and these have to be taken into consideration and reconciled in the presentation of findings.

Interpretive research has also been selected because it has a strong following in IT research (Rowlands, 2005, Åsvoll, 2013, Stahl, 2013). At the time of writing this thesis, Google scholar reveals that one of the landmark publications for conducting interpretive research in

IS specifically, namely Walsham (1995), has been cited a formidable 2112 times. With this abundance of literature comes a certain degree of confidence with regards to the rigor of the interpretive approach in an IS study. In addition, this track record also provides ample exposure of shortcomings that should be avoided. These shortcomings are revealed next.

3.3.6 Shortcomings of interpretive research

Dobson (2002) contends that a common shortcoming of interpretive research is that social structures are observed in isolation. He goes on to state that in order to conduct a more accurate inquiry into a social structure, the interaction of people, mechanisms, and practices must also be taken into consideration. In his opinion the researcher should appreciate the possibility of multiple direct or indirect influences acting on the social structure under investigation. To address this contention by Dobson, this study relies on the epistemology of Activity Theory (AT) (discussed in detail in Chapter Four) to ensure that the natural order of the social structure under investigation is preserved and a more holistic outlook is achieved.

Another potential shortcoming of interpretive research is its potential struggle to consolidate varying participant viewpoints (Smith, 2006). This line of argument is echoed in Dobson (2002) who suggests that, within interpretivist paradigms, reality and the understanding of this reality cannot exist independently of the social group that is responsible for constructing this reality. By implication, the understanding of reality will be different within different social groups. Hence, the inherent difficulty in "[…] the objectivity of data, reproducibility of experiments, or generalisability of results" (Stahl, 2013). Although it is difficult to completely eradicate this shortcoming from a research endeavor, the effects of this shortcoming were subdued in this study by employing more than one data collection technique, and by minimal researcher participation in certain project-based activities (Denzin and Lincoln, 2008, Yin, 2012). This allowed for closer interaction with the *social group* and hence, a better understanding and a more accurate subsequent portrayal of their reality. In addition, since the SMME case sites were largely the same size and had similar operational and management practices, there were some commonalities in their observed program management practices.

Having now discussed the first component of the research framework, namely the underpinning research philosophy; the second component of research strategy can be outlined.

3.4 A representation of the qualitative research strategy

This study incorporates a qualitative research strategy. Qualitative research typically consists of a framework that tries to translate the as-lived world into a series of representations, and may be defined as the immersion of the researcher into the environment under observation. Through the use of techniques such as field notes, interviews, conversations, photographs, recordings, observations, and notes to the researcher himself (Denzin and Lincoln, 2008), qualitative research generally attempts to illuminate the environment under investigation (Creswell, 2013). These techniques are asserted to be supported by the individual's personal experience, introspection, and historical and interactional artifacts which inscribe routine and meanings (Denzin, 2002, Denzin and Lincoln, 2008, Creswell, 2009, Silverman, 2010) and strongly influence daily activity and attitude towards the activity. In contrast to qualitative research, "a major disadvantage of quantitative research is that, as a general rule, many of the social and cultural aspects of organizations are lost or are treated in a superficial manner. The context is usually treated as noise […]" (Myers, 2013).

Common characteristics of qualitative studies are synthesized in Creswell (2013) and include data collection in the natural setting, whereby the researcher collects data himself without the aid of instruments designed by others; utilizes multiple techniques for data collection; or uses multiple modes of reasoning (e.g. induction, abduction) to establish codes and themes. The characteristics noted above aptly describe this research process. With the broad objective of making practices more visible as has been highlighted, qualitative research is deemed an appropriate and necessary research strategy.

The next section deals with case study methodology, which is the methodology that is selected to operationalize the research framework.

3.5 A definition of case study methodology

Case study research refers to an in-depth investigation of a particular instance of phenomena in their natural setting in order to generate results and create discussion around them (Lee, 1989, Stake, 1995, Rule and Vaughn, 2011, Yin, 2012); and it is perceived to be a widely used methodology for qualitative research (Remenyi, 2012). There are variations to the case study methodology that include explanatory, exploratory, descriptive (Yin, 2003), intrinsic, collective, and instrumental categories (Stake, 1995).

The case study methodology commissioned in this thesis best conforms to the descriptive category (Rule and Vaughn, 2011, Yin, 2011, Runeson et al., 2012) in that the aim is to generate theory, and not to test or apply existing theory. This is usually achieved by describing a sequence of events, underlying mechanisms, and relationships, in order to help the researcher in "[…] understanding the dynamics present within single settings" (Eisenhardt, 1989). With a descriptive case study, data is collected without disturbing the environment under investigation. The following quote provides an adequate definition of descriptive case study research:

> "A descriptive case study is one that is focused and detailed, in which propositions and questions about a phenomenon are carefully scrutinized and articulated at the outset. This articulation of what is already known about the phenomenon is called a descriptive theory. It helps to specify the boundaries of the case, and it contributes significantly to the rigor of the finished case study. The power and promise of a descriptive case study lie in its potential for mining for abstract interpretations of data and theory development. The main goal of the descriptive case study is to assess a sample in detail and in depth, based on an articulation of a descriptive theory. This theory must respect the depth and scope of the case under study, which is conveyed through robust propositions and questions" (Tobin, 2010).

This study also conforms to the multi-case variety of case study research (Benbasat et al., 1987) in that two software SMMEs formed the investigative cases; and this study best conforms to Linear-Analytic structure (Yin, 2009), whereby the case study report usually flows through a typical sequence of setting research objectives, reviewing relevant literature, electing methodology, data collection and analysis, and conclusion. In some instances of the literature, this is considered "[…] most advantageous when research colleagues or a thesis or dissertation committee comprise the main audience of the study" (Yin, 2009).

This thesis perceives case study research as being more akin to bringing out details of the phenomena under investigation, from the viewpoint of the participants, via multiple sources of data. This is aligned with the interpretivist nature of this research framework. The generic components of case study research, and the motivations for its use, as well as its potential shortcomings, are described in the following sections.

3.5.1 A description of generic components of case study research

The constituent parts of case study research will now be described, in order to elucidate how case study research will feature in this study. The following is a list of components of a typical qualitative case study:

a. Theoretical propositions
b. Research questions

c. Unit of analysis
d. Data collection techniques
e. Data analysis methodologies
f. Theory formulation
g. Validation of findings

(Rule and Vaughn, 2011, Remenyi, 2012, Yin, 2012).

For the first two components, earlier sections in Chapter Two suggested theoretical propositions, and a discussion was provided on the formulation of research questions earlier in this chapter. The unit of analysis component generally determines the choice of case study. Merriam (2002) suggests that qualitative case studies should be elected only when the case being investigated is unique or outstanding. In this thesis, the unit of analysis is software SMMEs that implement multiple simultaneous Scrum projects. These cases are considered unique due to their implementation of a relatively unknown nature of program management practices, as demonstrated by the literature survey.

Data collection, the fourth component, includes techniques for ethnography, observation, artifact analysis, and interviews. This approach is mirrored by some who profess the advantage of utilizing multiple sources of data, but "[…] without an agenda for what I might find" (Creswell, 2013). The data analysis component involved thematic analysis of transcribed fieldwork notes filtered through the lens of Activity Theory (discussed in Chapter Four). Data collection techniques and theory formulation are relatively intricate undertakings given the qualitative nature of this thesis. For this reason, they are discussed in more detail in dedicated sections later in this chapter.

Fitzgerald et al. (2006) maintain that software practice is often ahead of research; and state that: "the practitioner can contribute to developing, modifying or refuting the theory from the perspective of practice. This is particularly important in African contexts where theories are often imported from foreign contexts and applied uncritically […]" (Rule and Vaughn, 2011). It is the view of this researcher that more rigorous insights might be gained from careful examination of management practices in successful software SMMEs. The researcher is involved with lecturing and mentoring undergraduate student teams, and for summative assessment purposes, student teams are required to develop software systems for businesses and NGOs located in and around the CBD of the Eastern South African city of Durban. Despite easy accessibility to student projects, real world industrial projects are sought out because such projects incubate student teams from constraints like cost constraints and cross team resource sharing, and are essentially projects in a nurtured environment. Having now

described the generic components of case study research, motivation for use of the case study methodology is presented next.

3.5.2 Motivation for use of the case study methodology

Case study research has a proven track record in capturing knowledge in real life settings, and in the subsequent extrapolation and presentation of theoretical findings (Benbasat et al., 1987). In natural science, theory is usually established first and then tested; whereas case study research usually reverses this order (Lee, 1989). Furthermore, case study research is usually employed when the researcher has little or no control over the events occurring within the real-life context; and should be used to contribute to existing knowledge of "individual, group, organizational, social, political and related phenomena" (Yin, 2003), especially when *how* and *why* type questions are being posed (Rule and Vaughn, 2011). These depictions of case study traits complement both the research questions and the approach of this thesis, this being to explore the practices of Scrum teams as they go about delivering software systems for clients.

The literature reflects an abundance of case study reports in related research fields, including agile software development (Fitzgerald et al., 2006, Middleton and Joyce, 2010, Moe et al., 2010) and software management (Voss et al., 2002, Miettinen et al., 2010). Best practice guidelines are available in a plethora of publications, such as Merriam (2002), Yin (2012) and Creswell (2013). Nonetheless, case study research is not entirely devoid of potential pitfalls. Some of these are explored next, along with measures taken to help alleviate their effects in this research undertaking.

3.5.3 Potential shortfalls of the case study method

Case study research, like any other research approach, is not completely immune from criticism. Potential shortfalls and elaborations thereof, are presented in the sections that follow. These identified shortfalls stem mainly from five criticisms: the non-prescriptive methodological nature; the generalizing of findings; and challenges related to maintaining control of the observation process, deducing theory from findings, and lack of replicability.

First, case study research denounces a strict prescriptive methodological nature (Yin, 2003, Yin, 2012) which could negatively impact on the rigor in implementation. As a consequence, the researcher needs to be wary of biased findings and inaccurate conclusions. To cater for this potential shortfall best practice guidelines were examined, including literature that

identified IT research areas better suited for the case study approach (Benbasat et al., 1987); literature that proposed guidelines for avoiding case study research pitfalls (Lee, 1989, Yin, 2011); as well as literature that offered guiding principles to follow (Klein and Myers, 1999, Yin, 2012).

A second criticism refers to the difficulty in generalizing the results from case study research (Lee, 1989, Yin, 2003). However, some point out the "devastating" effects of the misconception of not being able to generalize from case study research (Flyvbjerg, 2006). This notion is supported by others who argue that "the aim of the case study is to generalize findings to theoretical propositions and not populations or universes" (Yin, 2003), and state that the goal of the case study "[…] will be to expand […] theories and not enumerate frequencies" (Yin, 2003). This notion is further supported by the following comment:

> "That knowledge cannot be formally generalized does not mean that it cannot enter into the collective process of knowledge accumulation in a given field or in a society. A purely descriptive, phenomenological case study without any attempt to generalize can certainly be of value in this process and has often helped cut a path toward scientific innovation" (Flyvbjerg, 2006).

Fitzgerald et al. (2006) proclaim that a richer picture may be obtained from even a solitary real life development context, and this view is supported by Mintzberg (1979), who asks:

> "What, for example, is wrong with samples of one? Why should researchers have to apologize for them? Should Piaget apologize for studying his own children, a physicist for splitting only one atom?"

Aligned with these arguments, this study does not strive towards generalization of findings. Instead, the aim is to contribute towards an emerging area of research.

A third criticism lies in the difficulty of maintaining a disciplined and controlled observation process (Lee, 1989). As so far as possible, the case study researcher needs to be unobtrusive in order to preserve real world observations (Yin, 2012). Hence, the researcher should reframe from a tight control mentality over the fieldwork, and instead allow the research to expand into unexpected areas of discovery that are not unlike the process of an unravelling murder investigation (Latour, 1987). Guidelines for conducting fieldwork that satisfies these requirements are provided in Forsythe (1999) and Schultze (2000) and both offerings were adhered to in this study.

The perceived difficulty in making accurate and logically linked deductions is a fourth criticism (Lee, 1989, Flyvbjerg, 2006). Some point out an apparent intrinsic difficulty in working with verbal propositions, and as a consequence relegate case study findings to sophisticated story telling (Diefenbach, 2009). Compared to propositions formulated in

natural science that can be proved more easily - either mathematically or statistically, making deductions from verbal propositions seems more difficult to validate. “How can we be sure that a qualitatively deduced finding is not, in fact, wrong?” asks Lee (1989). The author argues that in mathematical analysis rules can be used to check validity of deductions. These rules are based on logic. In the case of verbal propositions, there are no rules for deriving deductions; they are still based on logic, the same way mathematical rules are based on logic (Lee, 1989). Lee then cites the earlier work of Kaplan (1998) who states that even for Charles Darwin, logical deduction was based on verbal propositions and not mathematical rules because he could not reproduce the evolution of various species over the millennia.

The fifth and final criticism is the lack of replicability. Replicability of the case is not always possible when conducting investigations in a real life setting (Lee, 1989); each person, group of people, or organization under investigation is unique and will result in different findings. However, common themes can be extracted from different cases (Yin, 2012). In natural science research, by comparison, results of experiments are more easily replicated and hence, results are more easily generalized (Lee, 1989). Lee (1989) refers to this as nomethic research, where the primary aim is to test theories. Idiographic research, where results are difficult to replicate from one research setting to the next, is the opposite, and is a usual characteristic in case study research. Idiographic research is often used in studies such as this thesis that seek theory formulation, .

These shortfalls have the potential to pose serious implications for this study given that the case studies are the main conduits of this research framework. A review of literature on pragmatic experiences and literature by established research methodology authors, helped to provide a glossary of symptoms to avoid as well as a set of practical guidelines. These were adhered to in order to lessen the impact of the inevitable shortfalls presented above.

The case study methodology was supported by techniques for data collection and analysis, and a discussion on these comprises the fourth and final part of the research framework, which is expanded on next.

3.6 Techniques utilized for data collection

The next component of the research framework comprises techniques commissioned for data collection and data analysis. This section consists of two parts. The first showcases the elected data collection techniques. It begins with a deliberation on ethnographic concepts and how they informed observation, the interview technique, data coding, researcher

participation, and the period of time spent in the field. The second part deals with data analysis and introduces Activity Theory (AT) as the epistemological lens.

3.6.1 Crafting data collection through ethnographic guidelines

This study utilizes ethnographnic techniques in its data collection. Merriam (2002) argues that case study research is often supported and enhanced by other approaches like ethnography. That said, ethnography is also often defined as a research framework in its own right (Creswell, 2009). Although ethnography is commonly based on observational work within a certain environment (Denzin and Lincoln, 2008, Creswell, 2009, Silverman, 2010), Merriam (2002) cautions against thinking of ethnography and field work as synonymous. Ethnography highlights not how data will be collected, but rather how this data will be interpreted, that is, ethnographyß provides the lens for data analysis. Essentially ethnography must re-tell or re-create the situation under study for the reader. A similar viewpoint is that "learning to do ethnography involves learning to see social situations in a way that problematizes certain phenomena" (Forsythe, 1999).

Archer (1998) highlights the possible danger of flawed research emerging when a social system is closed or cut-off for the purposes of experimentation. Although immersed within the research environment, ethnography works best when maintaining a degree of separation between the researcher and the participants in their work environments (Schultze, 2000). Balancing both of these notions was deemed necessary in this thesis. On the one hand, the researcher could not influence the activities of the developers and management of the project, but at the same time, the researcher did participate in certain project activities whenever the opportunity presented itself. Furthermore, this researcher claims the usefulness of ethnography in uncovering and making explicit tacit knowledge. The next section examines guidelines and principles that should be adhered to in ethnography.

3.6.1.1 Ethnographic research guidelines and principles that were followed

There are numerous guidelines for conducting effective ethnography, ranging from earlier work by Patton (1990) and Schultze (2000), to current offerings by Silverman (2010) and Creswell (2013). The guidelines proposed by Forsythe (1999) were most appealing to this study. The first guideline stresses that the early stages of the fieldwork should include in-depth investigations into the knowledge work and the settings where this work is done. This is echoed in the assertion that "the most advanced form of understanding is achieved when researchers place themselves within the context being studied" (Flyvbjerg, 2006). Adhering

to the second guideline, this thesis did not strictly prescribe to early pre-determined research questions, knowing that these questions would most probably evolve as the fieldwork unfolded. This is in opposition to the positivist stance which sets up the experiment early on and relentlessly follows that plan throughout the study. The aforementioned ethnographic guideline is also aligned with the hermeneutic circle thinking, whereby the fieldwork can have an influence on theory formulation and vice versa.

The third guideline is the importance of filtering the data collected during interviews and observation, based on the premise that humans are not always able to accurately portray their daily tasks or elaborate on their thought processes (Forsythe, 1999). This study catered for this by having the researcher spend an extended period of time in the setting (the length of which is discussed below). Interviews with multiple participants triangulated responses.

The fourth guideline deals with the behavioural and organizational patterns which are mostly occasional and not always glaringly visible to the researcher. An interesting analogy offered by the author is that of a medical doctor performing a more accurate diagnosis from an x-ray of the patient, instead of a photograph (Forsythe, 1999). In line with this point of view, for this study careful analysis of the data is carried out using the unconventional elected lens of Activity Theory (AT).

In conjunction with the above guidelines, Forsythe (1999) further proposes three underpinning principles for conducting ethnography. For the first principle, consideration must be given to suitable techniques, such as observation, formal and informal interviewing, and viewing of artifacts. Secondly, it must be remembered that the techniques are usually influenced by the theory being used for data analysis. And thirdly, the application of the techniques are also considered to be influenced by the philosophical underpinning of the study, which Forsythe (1999) refers to as "maintaining epistemic discipline" (Forsythe, 1999). Following these three proposed principles, this thesis makes use of techniques such as semi-structured interviews, observation, and artifact analysis. Consideration was also given to the concepts and practices of AT used for data analysis, and finally, data collection was informed by the concepts of AT.

3.6.2 Time spent in field

In order to better understand and improve relations with a group of people within a given environment, the researcher should spend an extended period participating in activities and learning from the group (Silverman, 2010). Forsythe (1999) and Silverman (2010) stress that

it takes as long to train a good anthropologist as is does to train a medical doctor. This may be the case for Forsythe, due to her area of enquiry being a complex medical field with the people that were observed being doctors. In this study, the researcher is not completely alienated from the area of agile software development and management, having spent six years teaching related concepts to postgraduate students and four years mentoring undergraduate student Scrum software projects. An important constraint was the limited leave available to the researcher. Being a full-time lecturer, eight months was the total leave awarded before needing to return to duty. Hence, the time spent with the two software companies amounted to eight months.

3.6.3 Observation

Observation is defined as "the act of noting a phenomena in the field setting" (Creswell, 2013). Observation is a key tool for data collection in a qualitative study (Myers, 1997, Denzin and Lincoln, 2008, Creswell, 2013) and is often useful in portraying a sense of "being there" (Rule and Vaughn, 2011). Observation formed the primary data collection instrument in the study and included taking note of practices, conversations, and interactions. Bearing in mind the observer roles juxtaposed in Creswell (2009), this data collection mostly conformed to participant-as-observer. However, the role was not cast in stone and there were instances where the researcher was asked to participate in project activities such as innovation workshops and sprint retrospectives. When participating, the researcher was mindful not to influence the outcomes.

3.6.3.1 Observation guidelines that were followed

Observation conformed to the guidelines for observation presented in Merriam (2002), Silverman (2010), and Creswell (2013). These included determining which project team members and processes will be observed and for how long; the observer role; and the structure and method of recording observations. It also involved the immediate recording of observations. Observation notes were recorded using observation protocols suggested in Creswell (2013). In essence, this provided a template to give structure to that which was being observed. It included a heading, start- and end-time, descriptive notes chronicling the flow of observed activity, and reflective notes or researcher views on activities, processes, and summary information. Immersion into each of the SMME environments was carefully planned. The researcher introduced himself on the first day along with his purpose and intended duration of stay. Owner/managers of the companies briefed their staff before arrival

of the researcher. At first, observations lasted a few hours per day and this eventually grew to an entire working day. At the end of the observation period interviews were conducted.

3.6.4 Interview procedure

Interviews allow the researcher to seek clarity on ambiguous and difficult to comprehend data originating from other ethnographic techniques (Myers and Newman, 2007). They can afford the researcher a mechanism for eliciting opinion and comments from the individuals who form part of the research environment.

3.6.4.1 Guidelines adhered to for conducting interviews that were followed

Techniques and guidelines proposed by Fowler (2009) and Creswell (2013) were adhered to when conducting the interviews. Adhering to the first guideline, the purpose of the research was presented to each participant. He or she was then told that their involvement was voluntary and they could opt out at any point. They were told that any information that they might provide would be confidential. The projected duration and the procedure of the interview were also explained to the participant. Second, the questions asked were open-ended and the researcher only stated the contextual area for discussion via the use of prompts based on the program management knowledge areas identified during the literature review. Towards the end of an interview session, interviewees were encouraged to make any comments or express opinions on the research or express any other related opinions. Probing was the third technique of the interview. This occurred when the researcher was unclear on certain responses and required clarity, or when the response required decomposition. All interviews were recorded using a smart-phone. This is the fourth guideline proposed by Fowler (2009), who suggests that open-ended interviews be recorded verbatim. Respondents were assured that the recording was for the researcher's *ears only* and would not be published as a podcast or any similar technology. The fifth guideline was adhered to by the researcher being mindful of age, education, gender difference, and not presenting himself such that he appeared superior to the respondent. The final guideline aimed to overcome vocabulary and terminology barriers. This did not present a huge problem as all interviews were conducted in English and the terminology was well understood by both the participant and the researcher due to a shared familiarity with Scrum.

3.6.5 Transcribing and indexing fieldwork

Adherence to the data collection techniques described above resulted in a rich qualitative data set that was subjected to a thematic analysis. Similar to the procedure first described in Chapter Two (subsection 2.3.4), thematic analysis progressively moved from very broad categories to key themes. Key themes were distilled through the application of AT concepts (and this is discussed in Chapter Four). Yin (2011) suggests that within the context of a multi-case analysis, it is important to first identify themes within each case and then look for common themes across cases. This process was followed but subjected to structuring guidelines. For instance, not all information in a qualitative study is needed (Creswell, 2013) and codes should be assigned to recorded data based on the research questions (Runeson et al., 2012).

Observation notes and interviews were typed out in text format. Shorthand labels were adopted for indexing and referencing purposes e.g. *min_int_dev3_pg12:4* which translates to Minerva (researcher-given name to one of the case study sites), interview transcript, for developer number three (names are removed to preserve anonymity), page twelve, and paragraph four. This approach is sometimes called selective coding (Rule and Vaughn, 2011) where a code is a particular transcribed narrative. Later on, labels helped with cross-case code analysis. The grouping of similar codes within and across the two case studies resulted in a theme. Sometimes, single instances of codes were noted as interesting and relevant for theory development (Creswell, 2013). As suggested in Runeson et al. (2012), codes were not cast in stone; instead, they evolved throughout data analysis.

During the period of data analysis a visiting Professor at the researcher's place of employment held several workshops on Nvivo. Valid for 3 months per installation, the trial version was available for download from the Qsr International website (Qsrinternational, 2013). The full version was deemed too costly for a University wide licence. The trial version provides an organizing tool useful for indexing and cataloguing fieldwork data provided that field notes are first transcribed into text format. Nvivo was utilized to some extent during the data intensive phases of this study but tools can only assist and "[…] you must still be prepared to be the main analyst and to direct the tools; they are the assistant, not you" (Yin, 2009). Although tools like Nivivo may have helped with identifying similarities and common occurrences within the data, the process of deep analytical processing and theory building was purely a function of the researcher.

3.6.6 Researcher participation

The researcher attended formal and informal meetings whenever the invitation was extended to him. Meetings included sprint retrospectives, innovation workshops, sprint reviews, sprint planning sessions, sprint grooming sessions, and daily stand-up meetings. Although the researcher was present, participation and contributions in these meetings was guarded so as not to influence practices. Another aspect of concern during the early stages of fieldwork was that of alienation. At first developers did not see the researcher as part of the team and sometimes harboured the misconception that this was some form of audit and grading exercise. Participation helped to *break the ice* which, in turn, led to more free-spirited conversations and easy flowing informal communication. Towards the end of the field work period, and in both case studies, the majority of team members were enthusiastically helpful.

3.7 Theory formulation

The literature portrays theory formulation in qualitative discourse as an area of lively debate. For this thesis, highly cited publications by Whetten (1989), Weick (1989), Llewelyn (2003), and Gregor (2006) were closely examined for guidelines. As was the case with these authors, similar challenges manifested in this thesis. The important question that arises at this juncture is how best to structure and craft that which was observed into meaningful theory. This question is elaborated by Sewchurran and Brown (2011), who assert that "research contributions can be defined as the development and the use of theoretical concepts as a means to understand, observe, act, or bring forth a world with which to talk and interact" (Sewchurran and Brown, 2011); and Llewelyn (2003), who asks: "How should these actions and events be understood? How can organizational structures and processes be explained?" (Llewelyn, 2003). The next subsections investigate these questions further.

3.7.1 On taxonomies, frameworks, and guidelines for theory formulation

Llewelyn (2003) was examined first. Her contribution is twofold. First, the mainstay of this author's argument is the importance of theorizing, and recognizing that theory is part of every individual and plays a decisive role in their individual actions, communication, and reflection of practice. Second, the author proposes a framework of types (or forms) for contextualizing theory in five levels, that is tabulated in Table 7:

Table 7: Description of levels of theory based on Llewelyn (2003: 667)'s framework of types for contextualizing theory

Level of theory	Description
Level 1	Use of metaphors: The aim of this theory is to create a perspective of the world as experienced by humans. Individuals make use of metaphorical expressions in making sense of, and for communicating a meaning of, their world.
Level 2	Differentiation: Knowledge of the world is facilitated through pairings that serve as contradictions. Here, different sets of qualitative data may be categorized in opposition to each other.
Level 3	Concepts: The author suggests that concepts are central to theories of practice and are mostly responsible for the manner in which people react to and view the world. At this level, concepts are introduced and existing ones are refined to bring about new perspectives of the world. Level 3 is also described as the link between the lower and higher levels of theory. This is further elaborated on by the author who says "they create meaning and significance through linking the subjective and the objective realms of experience" (Llewelyn, 2003).
Level 4	Context Bound: At this level an attempt should be made to explain the social phenomena within their settings or environment. More specifically, to explain the social conditions under which practice or work takes place. The aim is to observe work practice not only as a technical endeavor, but also as practice within the given social setting.
Level 5	Context-free or Societal-level: The author cites the work of Habermas and Marx whose "grand theories" about social structures resulted in new perspectives and ways of thinking of society for example, modernism and postmodernism.

As a study progresses, theory may be constructed in an upwards manner from Level 1 through to Level 5 (Arnold, 2006); but bearing in mind that the researcher is not obliged to begin at Level 1. Instead, the chosen level depends on the characteristics of the study and the levels can be entered horizontally (Irvine and Deo, 2006) and then then progressed, represented as either an upward or downward movement. Carlsson (2011) suggests that Levels 1 through 4 are best suited for studies of traditional natural science or social science, while Level 5 might be more suitable for design science.

Tilson and Lyytinen (2005) provides an alternative suggestion that metaphor and concepts originate from the perspective of the researcher; they are the lens through which the researcher views the world. The metaphor is viewed in familiar terms that facilitate in communicating the study but do not necessarily form the foundation of all the other levels of theory. In line with Arnold (2006) and Carlsson (2011), this study will start at Level 1 and progress to Level 3 and eventually make a contribution through to Level 4.

Gregor (2006) is examined next and taxonomy of theory is outlined in Table 8:

Table 8: Gregor's taxonomy of theory (Gregor, 2006)

Type of Theory	Description
Analysis	Analysis and description of observed phenomena, without describing relationships between other observed phenomena.
Explanation	Theory attempts to provide explanations for observed phenomena but not to predict future occurrence. Propositions are not created for the purposes of generalization.
Prediction	Propositions and predictions are created but no causal explanations toward observable phenomena are required.
Explanation and prediction	Propositions, predictions, and causal explanations are provided.
Design and action	Theory provides explicit prescriptions for constructing an artifact.

Gregor's above-mentioned taxonomy differs from the levels proposed by Llewelyn (2003) in that it may be viewed as a set of pragmatic descriptions and not as a discussion on how to structure the theory development process. This study falls into the *explanation* category of Gregor's (2006) taxonomy noted above.

In summary then, the theory developed in this study aims to contribute towards an emerging body of knowledge. More specifically, the theory aims to show why program management, Scrum practices and SMME characteristics relate to and influence one another. This study will rely on the development of theory to relay its findings. This theory development process starts at Llewelyn's (2003) Level 1 (use of metaphors), progresses to Level 3 (concepts) and finally to Level 4 (context bound). The type of theory being developed falls into the explanation category of Gregor's (2006) taxonomy of theories.

3.7.2 Guidelines for theory development

Whetten (1989) proposes building blocks for the development of theory. These building blocks are presented in the form of questions designed to trigger critical thinking within the researcher; for example: What factors should be included to assist in the explanation of a certain phenomenon? Here the author suggests that the researcher strives to maintain a balance between relevant, unnecessary, or not-so-important factors. Another question is: How is the relationship between the factors identified? Here, an attempt should be made to establish the links, patterns, and causality between factors. The last two questions are simply: Why is this particular representation of the phenomena under investigation necessary? And, Will this new perspective contribute toward the field of inquiry?

A perspective of characteristics of *good* theory are synthesized in Rule and Vaughn (2011). These include the following: theory should exhibit laws of parsimony (that is, they must explain the most in the simplest way and lead to further research); theory should be coherent and falsifiable (theory could be proven wrong by contrary studies); and theory should explain phenomena. These guidelines will be adhered to in a litmus test fashion during the phase of theory development of this thesis.

The next section looks at the adopted approach for the generation of theory.

3.7.3 A definition of abduction and induction inference approach

The literature suggests that abduction was first introduced by Aristotle as a manner of logical inference and was advanced and popularised by Pierce in 1839 (Burch, 2010, Mingers, 2012, Ormerod, 2012). Abduction is a means of non-deductive inference for scientific research (Douven, 2011) and is often called inference to the best explanation (Douven, 2011b, Ormerod, 2012). To help elaborate by example; when the progress of a software project fell behind schedule (cause), the manager allocated more resources (rule), and the progress improved (effect). Although there may exist other causes or reasons for the manager's action, the hypothesis of being behind schedule is the most probable. Abduction seeks to identify a possible explanation for an observation. "Translated into interpretive case study research, abduction firstly plays the role of generating new ideas or hypotheses" (Åsvoll, 2013). The *firstly* refers to the initial step of most research undertakings, which is usually followed by evaluation of the hypothesis through induction (secondness) and finally empirically justifying the hypothesis (thirdness). Hence, abduction is sometimes seen as the first natural step in scientific enquiry; one whereby few facts are known at the beginning and the researcher has to fill in the proverbial blanks.

Abduction may be juxtaposed with induction and deduction. Induction may be defined as deriving a general rule from observations. More simply, induction as a means of theory generation, moves from the specific to the general (Ormerod, 2009, Rule and Vaughn, 2011). Using the example from above, induction identifies that the project fell behind schedule (cause) and when more resources where added (rule) the progress did eventually improve (effect). Hence one could infer inductively that all projects behind schedule will improve if more resources are added to them. Deduction, on the other hand, may be defined as deriving a conclusion from existing knowledge and observations (Douven, 2011b, Åsvoll, 2013). More simply, deduction as a means of theory testing, moves from the general to the more

specific (Ormerod, 2009, Rule and Vaughn, 2011), such as in the case of proving a hypothesis within a single research setting. By referring once again to the earlier example, the process of deduction identifies that the project fell behind schedule (cause). More resources are allocated to projects behind schedule to improve progress (rule). Hence one could infer deductively that the progress improved (effect).

Both abduction and induction are utilized in this study. Abduction is the preferred means for initial inference in this study. Due to program management practices being fluid and constantly shape-shifting, there were some unknowns during the early stages of this research undertaking. Using abduction and its method of most probable explanation, some practices were inferred. Once data collection was complete, inductive inference was employed for data analysis to validate those abductively inferred practices.

3.8 Summary of the chapter

This chapter first presented the main research question and subsidiary questions, before progressing to the intricate details of the research framework. The research framework consists of four major components; namely, the interpretive philosophical underpinning, the qualitative research strategy, case study methodology, techniques for data collection, and abductive and inductive reasoning approaches. Each of these components was explored in different subsections.

Interpretive philosophical underpinning is considered appropriate for this study due to its appreciation of individual perspectives and the need to capture the as-lived reality of participants in the fieldwork case sites. The qualitative strategy allowed for in-depth investigations of the two cases. This study did not aim to generalize findings, instead it sought to shed light on program management practices that seem immature and less established in the literature.

The chapter then highlighted descriptive, multi-site case study as the preferred methodology for operationalizing the research undertaking. A motivation for choosing the case study methodology was provided and potential shortcomings were exposed. Since observation was the primary data collection technique, ethnography best practices and principles were explored to guide its usage in this study. The constraint for the amount of time spent in the field was described, before moving on to a description of interviews. It was noted that interviews were conducted following guidelines presented in the discourse of ethnography. Observation and interview techniques generate a high volume of textual data. (In this study

interviews will consist of voice recordings which need to be transcribed). A coding and thematic classification scheme to catalogue fieldwork data and a coding structure for recording field notes were discussed. The researcher played the dual roles of research-as-participant and researcher-as-strict observer during data collection. The next subsection presented the practice guidelines for theory formulation. The chapter ended with the description of the preferred inference approach.

The diagram below provides an illustration of the research framework that is employed in this study, broken down into what was desk work and what was field work.

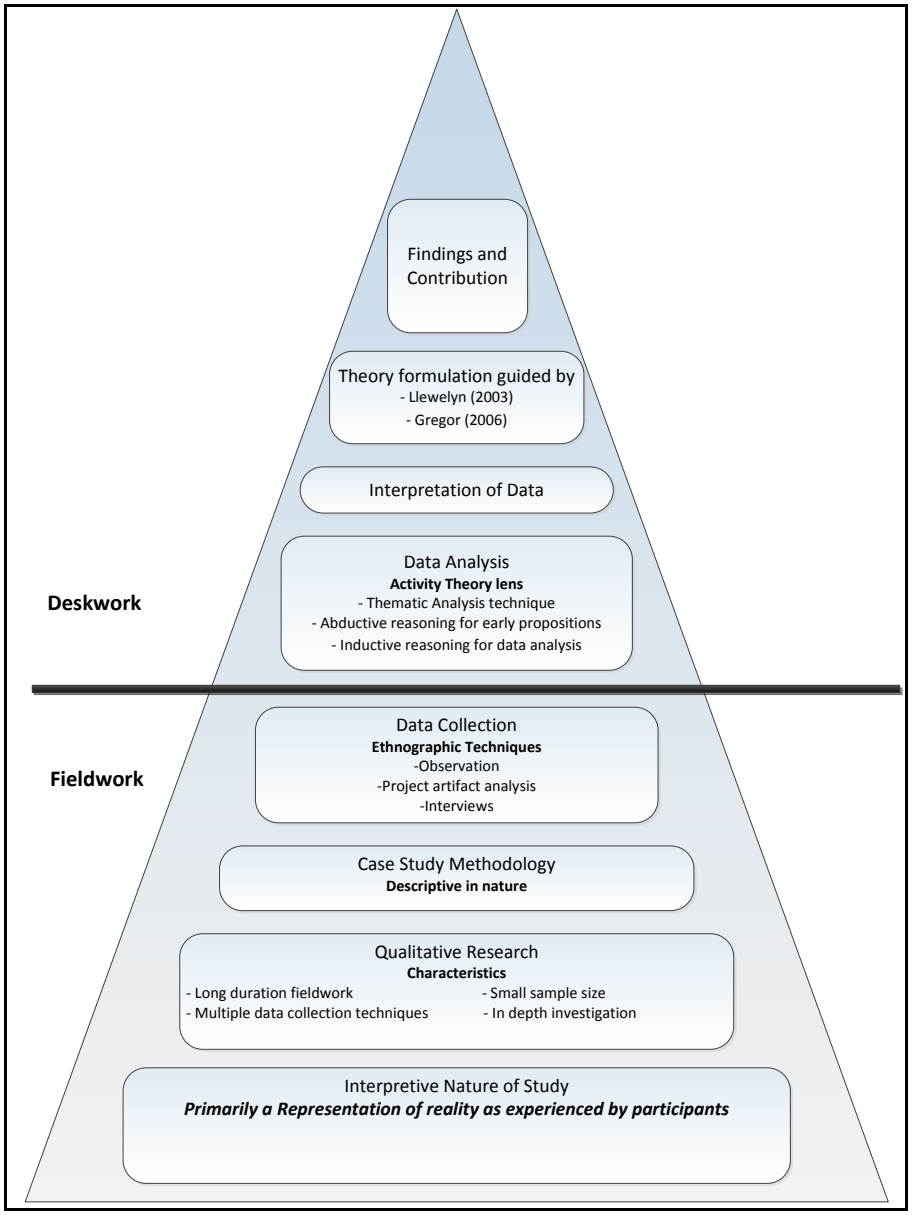


Figure 7: The study's research framework.
Source: Researcher's own construction

Chapter 4: A description of the data analysis methodology

4.1 Introduction

Activity Theory (AT) was the primary data analysis lens in this study. This chapter provides a detailed description of AT. This may seem like overkill on the surface but is deemed necessary by the researcher due the theory not being a conventional lens in IT-based studies. Given that this requires an extensive overview of AT, it was not included in the chapter on the research framework, for reasons of length.

The concepts of AT allow for a more pliable approach to elucidation of people, procedures, technologies, and any additional, unforeseen entities that might enmesh and form relationships as a particular software project progresses. A conference proceedings was published during the course of this study elucidating this notion; see Singh and Sewchurran (2014). The data analysis began with transcribing field notes. AT formed the lens for investigating practices for program management, and will be detailed in the sections that follow.

The chapter progresses as follows: first a definition of AT is determined, followed by a historical account that portrays the evolution of AT. The chapter then presents a view on the interface between the epistemology of AT and software projects before providing a description of AT's key components. The hierarchical view of operations, actions and activities is rendered; the zone of proximal development is describe; and the suitability of AT as a data analysis lens is explored. The chapter then portrays propositions of a Scrum software project and examines program management practices as AT activity systems. The chapter concludes with a review of IT-context-based studies that utilized AT.

The reader is to take note that Concept Analysis was a late edition to this study, commissioned to further refine the interpretation of data analysis in Chapter Seven. A full discussion of Concept Analysis is done in Chapter Seven.

4.2 A definition of Activity Theory

AT is often defined as a social-psychological theory for describing and better understanding human practices (Vygotsky, 1978, Kuutti, 1995, Crawford and Hasan, 2006); and it is sometimes referred to as Cultural-Historical Activity Theory (CHAT) (Engeström, 2000, Igira, 2012). In AT, activity is defined as a dialectic relationship between subject (an

individual performing actions) and object (a desired outcome) (Vygotsky, 1978). According to Crawford and Hasan (2006), "in this dynamic, purposeful relationship the 'always active' subject learns and grows while the object is interpreted and reinterpreted by the subject in the ongoing conduct of activity." Activity is viewed as the practical endeavor to achieve an outcome, and this outcome can be modeled using the interrelated components of an activity system (Engeström, 1993). In other words:

> "Practical activity is, therefore, a teleological notion based around purposes and goals. These purposes and goals are situated within an interpretative framework that constituents construct, participate in, maintain, and sometimes change" (Jarzabkowski, 2003).

For AT, human practice is perceived as a development process (Kuutti, 1995); that is, the practice itself is improved upon. The epistemology of AT posits technological and other artifacts such that they influence and are influenced by human activity. This is based on Vygotsky's theory of tools having a mediating influence on the interaction between individuals and objects (which is discussed in detail in section 4.5). AT should be seen as a framework of concepts that aim at providing a better understanding of relationships between interacting components that constitute an activity system. To fully apprehend the theoretical concepts of AT, an exploration of its historical development as well as its links with software projects, is deemed necessary.

4.3 A brief historical account of Activity Theory

Engeström (1987) portrays the evolution of AT through three phases of gestation. The story of the development of AT begins with Vygotsky's theory of mediated activity (Vygotsky, 1978). Vygotsky was a Russian born psychologist working on the educational development of children in the 1920s. In his model, human action was deemed to be influenced by society and culture, and was thus not deemed to be entirely the enterprise of an individual (Crawford and Hasan, 2006). In this triangular individual-society-culture model of mediation, actions of individuals are conditioned by the society in which they occur, since the society has an influence on the individuals: "The individual could no longer be understood without his or her cultural means; and the society could no longer be understood without the agency of individuals who use and produce artifacts" (Vygotsky, 1978). However, a shortcoming of this first version of AT was the predominant focus on individuals as the unit of analysis for AT (Engeström, 2001).

Vygotsky's student and a key proponent of AT, Leont'ev (1981), is often credited for generation two AT, that introduced the concept of differentiation between individual action and collective activity caused through historic division of labor (Engeström, 1987). This notion of collective activity instead of individual action, shifted focus away from the individual towards a focus on a community of interrelationships: "In activity theory, the distinction between individual goal-directed action and collective object-orientated activity is of central importance" (Engeström, 1999). Expansion of generation two of AT into Western academia started in the mid-1980s following the dismantling of the *iron curtain*. Sometimes referred to as Scandinavian AT, generation two AT combines the concepts of AT with Western developments in social and cognitive science (Engeström, 2008). Engeström is credited as the architect of the third and most recent generation of AT which unifies many of the components of the theory of activity by Vygotsky and Leont'ev. A new development in generation three AT is expansive learning (Engeström, 1999, Engeström, 2001, Engeström, 2009), where conceptual tools are employed to better understand "[…] dialogue, multiple perspectives and voices, and networks of interacting activity systems" (Engeström, 1987). A methodological framework for analyzing expansive learning is proposed in Engeström (2001) along with a case study demonstration of its use.

4.4 Establishing a link between the epistemology of Activity Theory and software projects

This section provides coverage of the key epistemological concepts of AT. These are tabulated in Table 9:

Table 9: Key epistemological concepts of Activity Theory

Concept	Brief Description
Unit of Analysis	The unit of analysis in AT is human activity (Kaptelinin and Nardi, 1997). Human activities are investigated within the context of their environment and are not recreated within artificial simulations or experiments (Kuutti, 1995). In addition, the focus of the investigation in AT is on the collective action and not entirely on actions taken by individuals (Kuutti, 1995, Nardi, 1996). This quote may clarify this: "[…] it is not possible to fully understand how people learn or work if the unit of study is the unaided individual with no access to other people or to artifacts for accomplishing the task at hand. Thus we are motivated to study context to understand relations among individuals, artifacts, and social groups" (Nardi, 1996).
Objective Reality	Individuals exist in a reality whereby *things* exhibit characteristics in natural science (e.g. colour, size, and cost) and in cultural social context (e.g. past experiences of usage) (Kaptelinin and Nardi, 1997).

Difference between Internal and External Activities	Internal activities are examined in unison with external activities (Kaptelinin and Nardi, 1997). Internalization is the conversion of external activities into internal ones. Internalization consists of, for example, imagining an alternative course of action or mental simulations. This process allows for contemplating alternative realities without actual interaction with real life artifacts (Kaptelinin and Nardi, 1997). Externalization is the reverse transformation; "externalization is often necessary when an internalized action needs to be 'repaired' or scaled" (Kaptelinin and Nardi, 1997).
History and Development	Activities are not static nor recursive. Instead, activities change characteristics and interfaces in a continuous developmental process. Furthermore, history in the form of lessons learnt and experiences, shape activities (Kuutti, 1995).
Artifacts and Mediation	Completion of activities usually rely on artifacts like operational policies, technologies, and a plan. These artifacts play a mediating role. The relationships between the components of an activity system are not direct but a mediated influence. Artifacts are created and/or transformed during the course of the activity and are also influenced by history (Nardi, 1996). This idea was first established by Vygotsky and is represented in his basic model of human activity (Vygotsky, 1978).

In this study the epistemology, or the theoretical assumption of the structure of reality, follows the contours of AT. To elaborate very briefly, there is a natural symbiosis between the epistemology of AT and the investigative requirements of this study for the following reasons: First, software development work is a team endeavor and the collective thought processes and associated practices have to be considered. Second, software development within a SMME environment is foreseen as an environment that is heavily influenced by so called mediating factors, including the likes of limited resources and changing client needs. The epistemology of mediating artifacts appreciates these relationships. Third and finally, program management practice in the context of agile is far from a mature field of enquiry. To this end it is envisaged that observed practices will exist in a high state of evolution or at least in a state of trial and error. These insights are revisited in much more detail in a later section.

4.5 Key components and the basic structure of Activity Theory

For AT, an activity is the process of transforming an object into a desirable outcome. Objects may be viewed as both tangible items, for instance software requirements; or intangible items such as an idea which is "shared for manipulation and transformation by the participants of the activity" (Kuutti, 1995). Subjects are actors or individuals carrying out the object, or stated more simply, the objective (Mwanza, 2002). AT theorizes that this is not a straightforward transformational process, but is instead one that is arbitrated by a third component simply called an instrument. Instruments enshrine knowledge about certain

activities and can mature and change through usage. Hence, instruments embody history and social perspectives and serve as either enabler or inhibitor to the subject-object relationship. Figure 8 illustrates this relationship.

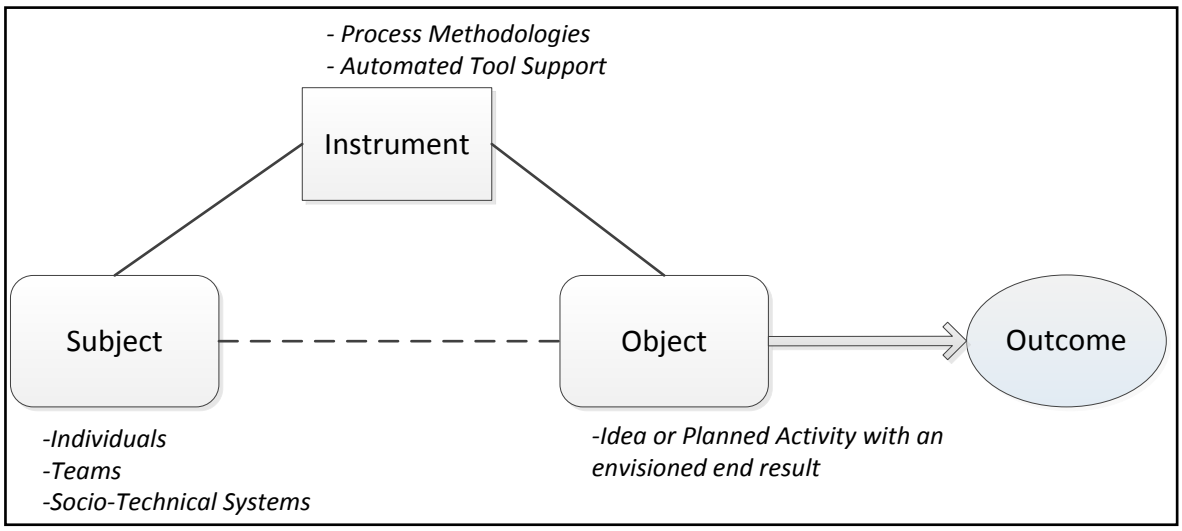


Figure 8: Mediated transformation process.
Derived from Kuutti (1995).

However, the above conceptualization of a transformation process in early generations of AT was in some cases perceived to be "too simple to fulfill the needs of a consideration of systemic relations between an individual and his environment in an activity […]" (Kuutti, 1995). Engeström later supplemented this conceptualization of a transformation process by inducing components which allowed for scrutiny of the activity, while appreciating that it is embedded within some social context (Engeström, 1987). Hence, additional components were added to provide a more systemic or holistic interpretation of the activity system.

This more recent version is illustrated in Figure 9:

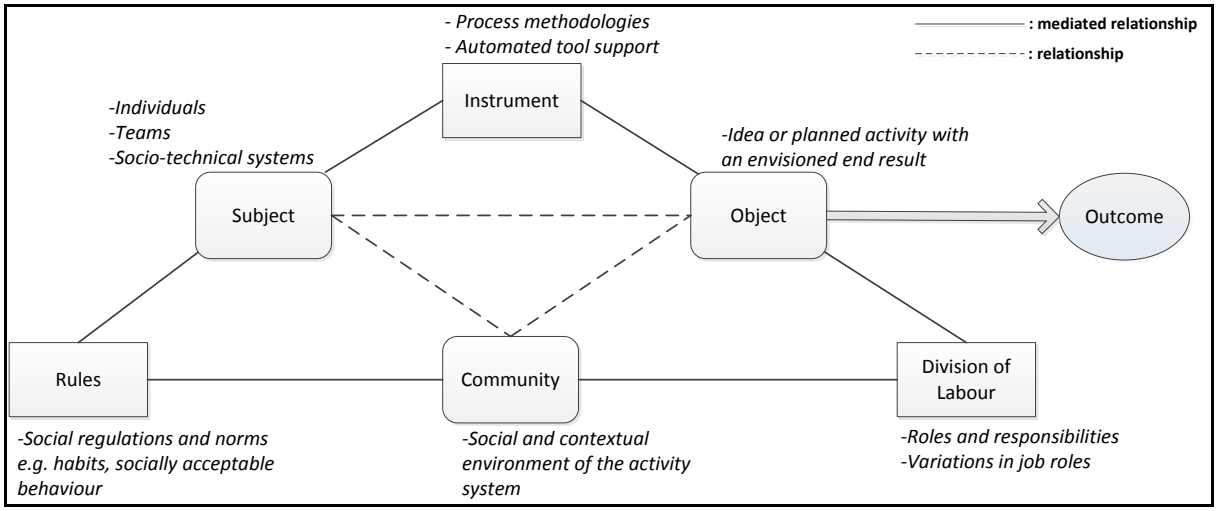


Figure 9: Addition of community component with associated rules and division of labor mediators.
Derived from Engeström (2008).

In Figure 9, community is the third major additional component that is added. The incorporation of the community component forges two new relationships, namely subject-community and object-community. Abiding by the epistemology of mediating influences, rules and division of labor are introduced as mediators for these relationships. Rules are the accepted and established norms for carrying out work within the community. Division of labor refers to how the community is organized in order to achieve the outcome. Both rules and division of labor can exist in tacit or explicit forms. Each of the mediating components, namely instruments, rules, and division of labor, is usually shaped through historical context and can change (Kuutti, 1995). Figure 9 also points out Engeström's concept of three reciprocated relationships between subject, object, and community. The social and cultural DNA of the company is embodied in the community component. As the social and cultural norms change, so too does its influence on activity. This notion supports the epistemology of the changing nature of an activity system.

4.6 Hierarchy of activity

Activities result in outcomes through a series of steps or phases and are generally too long to be completed at once. To cater for this, activity is viewed in levels of hierarchy; namely activity, action, and operation (Engeström, 1987). These are referred to as chains of action (Kuutti, 1995). Figure 10 shows this conceptualization using a single software project perspective:

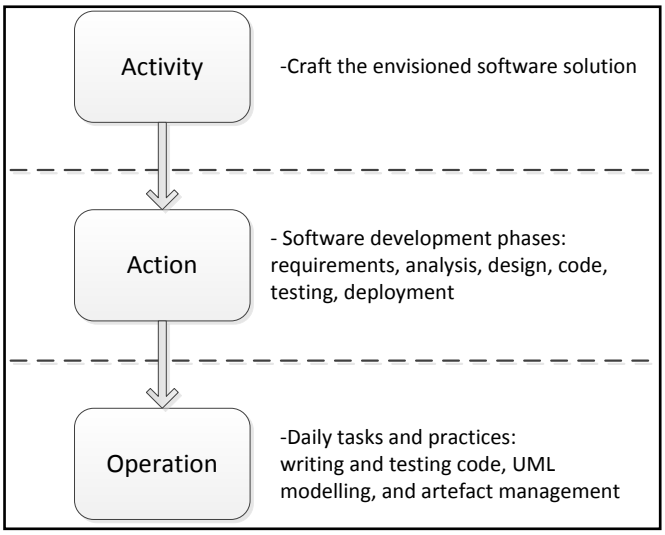


Figure 10: Hierarchy of an activity.
Derived from Kuutti (1995).

Activities are accomplished via chains of interactions between actors working towards the same outcome. Actors participating in an activity perform actions that have a short-term, well-defined goal. This action is situated in context and cannot be understood in isolation. The same action can belong to a different activity but might have a different meaning in that activity's context. Actions, in turn, are made up of strings of operations (Engeström, 1987). Operations are well defined routines used to navigate a course of action in response to changing conditions in the activity system. Operations are considered more fluent than actions (Nardi, 1996). Stated differently, when "[…] the action has been practiced long enough, the orientation phase will fade and the action will be collapsed into an operation, which is much more fluent" (Kuutti, 1995).

The hierarchy is not always as straight forward as is portrayed above. For example: "[…] there are no firm borders: a software project may be an activity for the team members, but the executive manager of the software company may see each of the projects as actions within his or her real activity at the level of the firm" (Kuutti, 1995).

4.7 Understanding change within the activity system: Zone of proximal development

AT activity systems are perceived as constantly changing (Engeström, 1999), primarily due to interactions and relationships between components, and with other activity systems. Contradictions within the activity system are mostly triggered by external influences (Kuutti, 1995, Allen et al., 2013) and can manifest in the form of problems, clashes, and breakdowns.

AT posits contradictions as catalysts for both innovation and change as the activity systems work towards overcoming their challenges (Engeström, 2000).

The zone of proximal development is a concept unique to AT. "The zone of proximal development defines those functions that will mature tomorrow but are currently in an embryonic state, i.e., the 'buds' of development." (Engeström, 1987). Activity systems pass through these zones of proximal development as they undergo transformation caused by constant ambivalence and contradiction. "In activity theory, developmental transformations are seen as attempts to reorganize, or re-mediate, the activity system in order to resolve its pressing inner contradictions" (Engeström, 1999). Examples may materialize in the form of experience of individuals and lessons learned from past activities that can cause a shift in procedural norms and operations. Table 10 presents a summary of key concepts in AT.

Table 10: Key concepts and components of Activity Theory

AT Concept	Description
Activity system	A collective formation of interrelationships and mediations.
Action	Chains of operations that lead to an action, that is usually driven by a well-defined, short-term goal.
Operation	Chain of short tasks that collectively lead to action.
Subject	Individual or collaborating actors.
Object	The purpose of the activity system.
Instruments	Physical and conceptual tools that mediate the relationship between subject and object.
Community	Social and cultural context (or environment) of the activity system.
Rules	Social regulations and norms for carrying out work that mediate the relationship between subject and community.
Division of Labor	Allocation of responsibilities and variations in job roles that mediate the relationship between community and object.
Zone of proximal development	Change or transformation of an activity system in response to pressing internal contradictions. An activity system will pass through several zones of proximal development in its quest for a desirable state and to produce an desirable outcome.
Contradiction	Serves as catalyst for transformation within activity system via innovation.

The next section explores the suitability of AT as a data analysis methodology.

4.8 Is Activity Theory a suitable data analysis lens for uncovering program management practices?

This thesis believes AT to be a suitable theory for describing emergent program management practices. Reasons for this standpoint can now be outlined. The epistemology of AT allows for exploration and discovery of influencing relationships between the three core entities of this thesis, namely Scrum, software SMMEs, and program management practices. These core entities metaphorically represent interlacing spheres rather than supporting tiers. Scrum project characteristics and SMME management practices are dynamic and often change when facing signs of distress. This thesis claims that AT is a methodology attuned to cater for this dynamism, largely due to its epistemology of mediating components that are situated within a specific context.

The first potentially useful AT concept is that of mediating relationships. Successful agile software projects, including Scrum, depend heavily on underlying principles such as intense collaboration between stakeholders (like the Product Owner and the development team members). In addition to this, Scrum also accommodates change in client business needs. As businesses respond to economic demands, Scrum triggers a ripple effect change in functionality of the software. Allowing for this dynamism in concurrent software projects induces strain. Not only can these changes to functionality be seen as external mediating influences, but as contradictions as well.

Program management in the context of a Scrum software environment, is not envisioned as a neat, linear sequence of activities, but is more akin to chains of interactions, growing in size and nature as a project progresses. AT concepts like zone of proximal development and positioning human and non-human actors on equal footing, cater for exploration of this liquefied sociotechnical network. With Engeström's influence and its subsequent evolution, AT is now better armed to explore dynamic activities within a specific social context. This is vital in light of Scrum practices exhibiting a less prescriptive demeanor and being more akin to reflexivity and fluidity. Hence, although the two case studies elected for this study may exhibit differing manifestations of Scrum and associated management practices, AT will allow the researcher to reconnoitre similarities and differences within these practices.

Software development is a complex undertaking that allies a plethora of human and technological or *non-human* actors on an equal footing. Actors may include people, methodologies, innovation, ideas, software and communications technology, as well as a

range of automated project management support tools. Using the concept of an activity system, AT acknowledges both human and non-human actors, and can facilitate in the description of their influential and symbiotic relationships. Furthermore, whilst culture is a tacit, difficult to measure concept, it is nonetheless one that is dealt a hand in shaping practices in software development environments. Although culture is usually an emergent trait and one that varies greatly from one SMME to the next, or even from one development team to the next, it has to be considered. AT has conceptual components such as community that allows for the incorporation of *hazy* concepts like culture. According to Lyytinen and Ngwenyama (1992):

> "The traditional view of work which underpins much of the research in computing and information systems is founded on a mechanistic "theory of work". It suggests that work can be divided into a sequence of tasks characterized solely in terms of their "uncertainty" and the need for "information" to make "decisions" (Lyytinen and Ngwenyama, 1992).

The authors go on to stress that a mechanistic outlook of work mostly ignores its multidimensional nature, and often results in an over simplistic understanding. It is the belief of this thesis that AT allows for the description and analysis of change within the activity system; change is viewed as a potent feature in the context of this thesis as agile projects embrace change, and the software SMME typically exhibits a shape shifting nature. In other words, in agile software projects, change is a natural occurrence and overarching program management practices are mostly unstable.

Given the points of motivation above, at a theoretical level AT is seen as a suitable lens for data analysis. The next section provides a propositional representation of a Scrum software project through the AT lens.

4.9 Possible representation of a typical Scrum software project using Activity Theory concepts

Software developers, Scrum Master, Product Owner, and owner-manager are the main *subjects* embroiled within a typical Scrum software project. The *object* may be represented as a yet to be delivered software system. Scrum software projects entail human and non-human actors in concert progressing towards delivery of a software system collectively forming the *community*. At the project level there is constant interaction between client liaison, developers, Scrum Master, Product Owner, and owner manager; as well as between mediating *instruments* such as product backlog, release plan, project management support

software, and burndown and velocity charts. Mediating instruments may hinder or support the major endeavor of achieving the desired project outcome.

Challenges may manifest in a multitude of formats during the course of a software project. Misunderstood or missing requirements details from clients, new technologies, or regression faults or bugs, are but a handful of contradictions that will trigger a rethink and resultant shift in project planning and subsequent operations. An example of such contradictions includes time required for the development team to learn a new development technology against the planned delivery schedule, or perhaps changing client requirements versus the allocated budget for the project. Scrum conventions can be viewed as the *rules* associated with the community. The organization of resources and actors adheres to guidelines stipulated by Scrum, although these are often tailored for better fit to the team and project. This is representative of the *division of labor* component that plays a mediating role for the community of actors and the object. The team participates in a periodic process of reflection via sprint retrospectives that allows the team to make note of failures and successes in their practices, for the purpose avoiding these in future. Practices such as these satisfy the socio-historical perspective of the community.

4.10 Possible Activity Theory perspective of program management practices

Having delineated the nature of activities within the project space, what of the practices in the program management space? The project level space of software development is often considered to be one of high innovation and dynamism. The overarching program management level has to exhibit similar characteristics. This is necessary for the constant striving for balancing resources and meeting project objectives. Changes to project organization, resource allocation and objectives, forces resolution of contradictions in the program management space as the activity system passes through zones of proximal development. In addition, the characteristics of SMMEs, as well as a need to conform to Scrum regulations, also influence the nature of program management practices that manifest in the form of rules, division of labor, and instruments.

In this thesis, program management practice within the context of SMMEs implementing Scrum is the activity system of interest. Figure 11 uses AT to model possible constituents of this activity system.

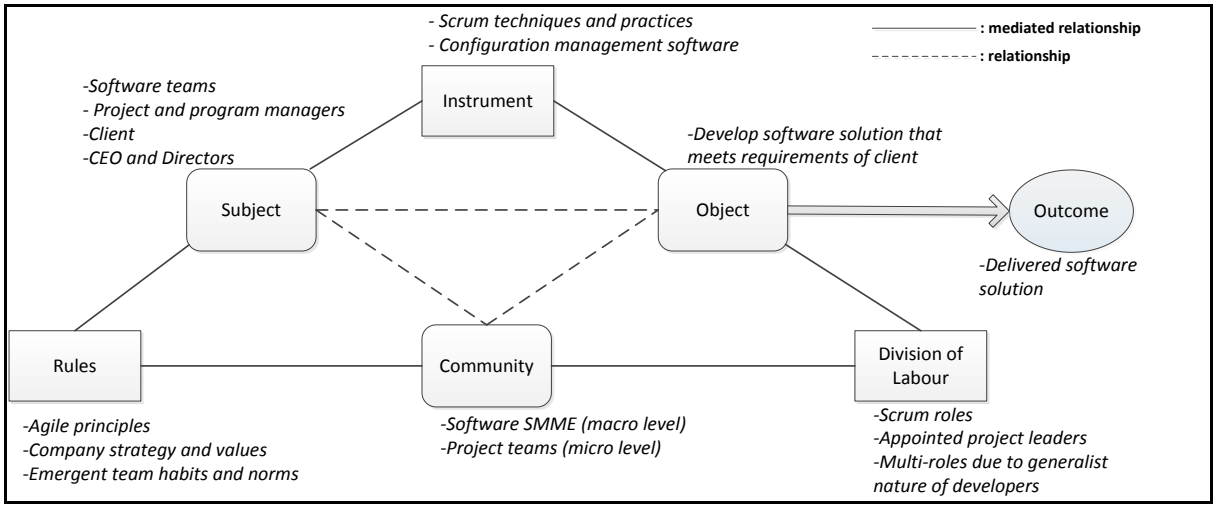


Figure 11: Possible Scrum program management modeled as activity system. Source: Researcher's own creation.

Having rendered an AT proposition of practice at the project and overarching program management levels, the next subsection provides a brief synopsis of some IT-related studies that have utilized AT.

4.11 Review of studies that utilized Activity Theory

AT features in a variety of areas, such as Human Computer Interaction, Information System Development, and IT strategy and implementation. Good commentary of this wide range is provided in Igira and Gregory (2009). Some of the AT developments include a stepwise model for operationalizing the concepts of AT to better understand computing systems (Mwanza, 2001, Mwanza, 2002); ISD as a collective work activity (Korpela et al., 2002); and strategic management practices (Jarzabkowski, 2003); and a group of researchers aim to advance the field of HCI via AT (Kuutti, 1995, Nardi, 1996, Kaptelinin and Nardi, 2012). Closer to the context of this thesis, McNely et al. (2012) employ AT to trace the activities of a student group working towards completion of an Scrum assessment project. Their research focused on project level practices. Recent developments have used AT as an underpinning for proposed research models aimed at better understanding IS design, implementation, and use from a social and organizational perspective (Igira, 2012); and to elucidate the process of IS development, process improvement, and enterprise architecting (Luukkonen and Mykkänen, 2012).

This subsection was not devised as an exhaustive list; instead it aims to showcase the penetration of AT into related research areas.

4.12 Chapter Summary

This chapter introduced Activity Theory (AT) as the data analysis lens for this study. AT was defined as a framework of concepts designed to elucidate the relationships between the objects that make up a human activity system. An activity system is a collection of interacting individuals and non-human objects that are striving to achieve some desirable outcome. An important aspect of AT is that the influence of objects, and the relationships between objects, can either hinder or support the objectives of the activity system.

A depiction of the evolution of AT was given to help clarify its current theoretical shape and structure. Key epistemological concepts of AT were described next, along with reasons for why these concepts can help realize the aims of this study. This was followed by a discussion on the major components of AT along with the relationships between the components. The version of AT revised by Engeström is utilized in this study.

The chapter also provided a hierarchical view of operations, actions and activities conceptualized in AT. This elaboration was necessary to show that program management practices may have different levels of hierarchy depending on the employee performing the activity and that the hierarchy may change over time as the activity system matures. The zone of proximal development is an important theoretical concept in AT in that activity systems are perceived as constantly changing in nature as they respond to influences from components and from interference from outside the activity system. Given the typical dynamic nature of agile software development practices, this zone of proximal development was deemed a useful concept.

The chapter then questioned the suitability of AT as a data analysis lens. This thesis posits that AT is indeed suitable and motivations for this standpoint were provided. Propositions of a Scrum software project and software project program management practices were then presented via the lens of AT. The chapter concluded with a brief survey of studies that utilized AT. This was done to show the penetration of AT into IT research areas.

Having now concluded a discussion of the research methodology component of the research framework, Chapter 5 will introduce the two case studies in detail and provide a broad overview of the fieldwork.

CHAPTER 5: FIELD WORK

5.1 Introduction

This thesis declares that software development and overarching program management represent a set of complex practices and these were examined in a natural, real life setting rather than an artificially created environment. To satisfy this need, two software SMMEs formed the units of analysis. Both SMMEs practiced a variation of Scrum and true to the form of characteristics of software SMMEs, both exhibited multi-project modes of operation. They were of similar size and exhibited a likeness in their organizational organograms. The major difference between the two SMMEs lies in their differing years of trade. Both SMMEs are situated in the eastern region of South Africa. To protect the identities of the SMMEs, pseudonyms are used, namely Athena and Minerva, to honour a confidentiality clause signed with both SMMEs.

Through the trio of main data collection techniques - namely observation, project artifact analysis and interviews, the activities and actors participating in and contributing towards the shape of program management were documented. The epistemology of Activity Theory (AT) focused the image of program management portrayed in this thesis. The ontology of the thesis is interpretivism; hence the preservations of properties (such as the differing perspectives of individuals), multiple sources of data, and investigation within a natural setting, were carefully considered.

The rest of this chapter begins by describing the case sites, Athena and Minerva and providing an organizational organogram for each. This is followed by an overview of the fieldwork conducted in both case study sites, before concluding with a summary.

5.2 Overview of the fieldwork

The case study research method was used in both SMME settings. Chapter Three provided a detailed discussion on the research framework including the case study method, and Chapter Four described the study's epistemology of AT. This section describes the data collection procedure that was utilized.

The fieldwork for both case studies took place at the SMME sites. Both SMMEs are located in a suburb approximately 20 kilometers outside of the city of Durban, which is situated in the eastern midlands of the province of KwaZulu Natal, South Africa. Working days at

Athena and Minerva were Monday to Friday, and if necessary some developers used a few hours on Saturdays and public holidays to *catch up* if the need arose. A typical day in the field started at 09h00 and ended between 16h00 and 17h00 for both case sites.

Before commencing the field work, meetings were held with the CEOs to discuss the aims and needs of the study. Some personnel from both Athena and Minerva have presented guest lectures, conducted student training sessions and serve on the industry advisory board at the researcher's affiliate academic department. As a result, there was a prior degree of interaction between the researcher and these SMMEs, albeit a small amount. One week after these meetings, fieldwork began at Athena. The CEO made introductions and over the first few days the researcher got to know the developers. Being a SMME with open plan offices, informal communication was easy to undertake; the researcher therefore got to know all the employees fairly quickly. The same introductory procedure followed at Minerva once the fieldwork at Athena reached completion.

Data collection primarily involved observation and interviews. Secondary data included items such as photographs of artifacts (such as Kanban boards), project documentation, documentation of informal communications, minutes of meetings, and outcomes of workshops. The researcher was given a desk within the open plan office at Athena and a desk within the same office of the Scrum Master at Minerva. In both cases, the researcher was free to roam the open plan office and was provided ample opportunity to observe and question practices. Interviews were conducted towards the end of the fieldwork for both Athena and Minerva.

The next section describes the case sites, Athena and Minerva, in more detail.

5.3 Athena: The first software SMME case site

The following sections provide cursory details of the first case study site, Athena.

5.3.1 Background

Athena is a South African owned and run software development SMME. It has been in operation since the turn of the millennium. Its clientele consist of local, national, and international businesses. Personnel often travel to clients for initial requirements and business analysis meetings. Athena was the recent recipient of a prestigious award, which only four companies in South Africa have ever received.

At Athena, the guiding ethos was the importance of providing value for a client, and not simply providing a software system.

> "If it is not solving a business problem then that's not the project we will engage in. We need to understand what business value a software system will add and the value propositions must be well defined" ath_int_ceo_pg1:3.

Athena's core business is software solutions consulting and software development. The software developed might be a brand new product built from the ground up, bespoke, or an interface with the client's existing software systems. On-going maintenance and upgrades to existing products were sometimes commissioned. Athena had a handful of in-house products that were sold as off-the-shelf products, such as ERP software systems. These software products existed in the form of long-term projects and underwent continuous development and upgrades. Dedicated staff were assigned responsibility for their upkeep. Other services that Athena provided included training for Agile and Lean development, and guest lecturing at local and national universities.

Athena employed a Scrum software management framework and its operations were based on guidelines for agile. Most of the software was developed using Microsoft technologies including platforms, and client and web technologies. Athena strived to involve its clients closely in the development process. For the duration of the fieldwork, Athena had a liaison person from a client present in the open plan office for one of their projects.

5.3.2 Employee organogram

The personnel at Athena included the CEO (founder), Operations Director (co-founder), Technical Director (partner), and 12 developers comprised of user experience designers, testers, analysts, and programmers. Two developers were employed on a contract basis. These developers may be described as freelance employees. The major roles of these participants are described in Table 11 and an organogram is provided in Figure 12. That said, as is characteristic of SMMEs, these responsibilities were not cast in stone and it was not unusual for developers to navigate to roles outside their area of expertise.

Table 11: Description of employee roles and responsibilities at Athena

Role	Responsibility
CEO	Responsible for the overall planning of business operations, strategic planning, and finances.
Operations Director	Oversees innovation and efficiency in the software development and project practice. This employee was responsible for coaching the development team on agile and Scrum practice and guidelines.

Technical Director	In charge of the development technologies used in software production. An adept developer, this employee explored innovations and new methodologies and attempted to integrate them into software projects. This was an attempt to streamline production time in projects via successful introduction of state-of-the-art technologies.
Scrum Master	Oversees project schedule, budget, and performance monitoring. He was responsible for resource allocation between concurrent projects and reporting on project progression to personnel and when required, also to the client.
Developers	This grouping of employees was the workforce of Athena. Their primary role was to write code for the various software projects. This grouping consisted of senior developers, testers, analysts, database developers, and user experience designers. Developers were employed on a full-time or on a contract basis.

The organogram below depicts the hierarchy of employees at Athena.

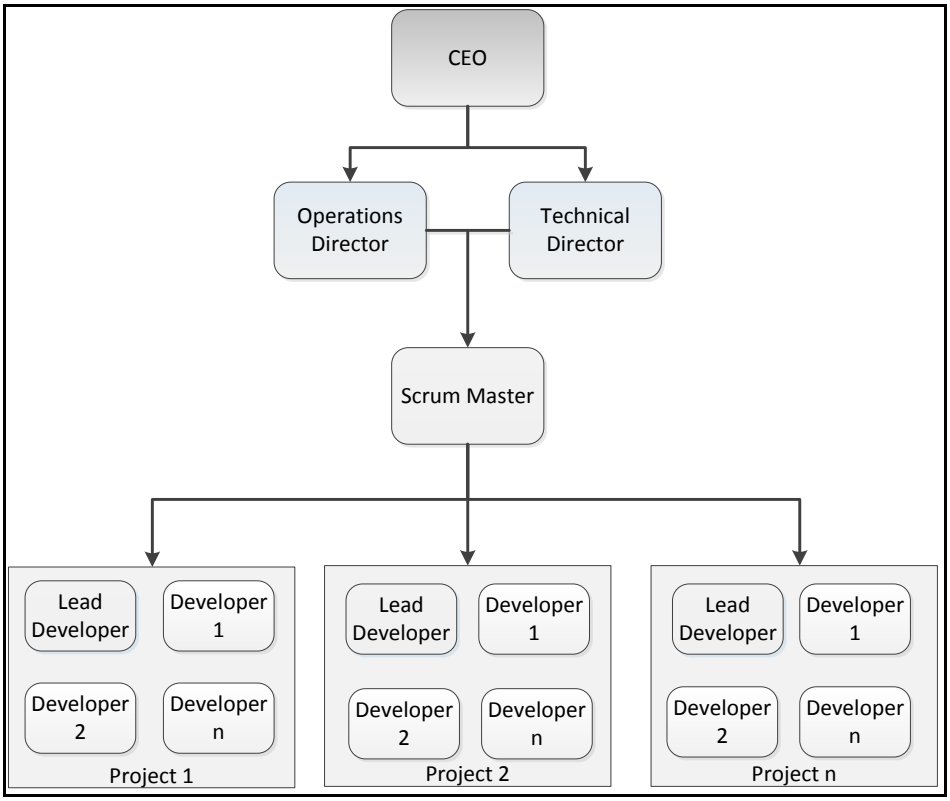


Figure 12: Athena employee organogram.
Source: Researcher's own construction.

The next subsection focuses on the second case study site, Minerva.

5.4 Minerva: The second software SMME case site

The following sections provide a synopsis of the second software SMME case site, Minerva.

5.4.1 Background

Minerva is a South African owned software development SMME and was founded in 2004. Its original branch lies in the same suburb in KwaZulu Natal as Athena, and it also has a new branch becoming operational in the city of Cape Town, in the Western Cape. The SMME started off by building simple websites for clients, and then slowly graduated to more complex database-driven and project management websites. From that point onwards, impetus evolved to include web-based software systems for a few clients, which included mobile development when that area of development was still emerging. Minerva perceives itself as a company that thrives on providing technological innovation and cutting edge software solutions for business challenges.

> "From the beginning our vision has always been to be an incubator for products not a consultant company. Consulting custom software development products for clients was really just a means to an end to generate revenue to fund product development" min_int_ceo_pg1:1.

At the time of writing, Minerva specialized in design and development of web- and mobile-based software products for local, national, and international clients. Apart from these client software solutions, home grown products were also developed and sold as off-the-shelf software, including support and maintenance. Both client software and home grown software existed as either long-term or on-going projects. Some client software had been advancing for more than two years at the time of fieldwork.

> "[…] there are no predefined requirements up front. Instead, we have freedom to innovate on their system. We have also gained their trust over the two years and now they [the client] have a dedicated budget for us" min_sm_obs_pg3:3.

At the time of conducting fieldwork, Minerva was managing seven concurrent projects at the case study site. Minerva mostly employed agile principles in its software development and management practices. A variation of Scrum was practiced.

> "We don't use the traditional Scrum, but a variation of it. We find that traditional Scrum works well for internal projects" min_sm_obs_pg1:2.

Minerva supported close client collaboration for the duration of projects and group telephonic conversations with the different clients was frequently observed.

5.4.2 Employee organogram

Similar to the case of Athena, the roles at Minerva were not cast in stone with personnel *wearing many hats*. This is in keeping with the generalist nature of employees in a SMME. The personnel involved in the case at Minerva included the Commercial Director (founder), two Technical Leads (one a founder and the other a partner), Project Manager (partner), Scrum Master, and 13 developers comprised of user experience designers, testers, database developers, and senior developers. Most startling is the absence of the role of CEO. The major roles of these participants are described in Table 12 and an organogram is provided in Figure 13.

Table 12: Description of employee roles and responsibilities at Minerva

Role	Responsibility
Commercial Director	Mostly responsible for crafting and implementing the business strategy and development. He was also responsible for first contact with clients and creating proposals for software systems. When asked to state his responsibilities the response was: In keeping with the owner/manager duality typical of SMMEs, the Commercial Director also got involved with technical work, such as architecture and design solutions. He had a background as a developer specializing in user experience design.
Technical Director	Two employees had this role. They both researched and implemented state-of-the-art software development technologies and practices in software projects. They were also responsible for designing and in some cases developing software architectures that serve as foundations for other software projects.
Project Manager	This employee was responsible for Business Development and overseeing longer term projects.
Scrum Master	Oversaw project schedule, budget, and performance monitoring. He allocated resources between concurrent projects and reported on project progression to personnel and, when required, also to the client.
Developers	Comprised of testers, user interface designers, business analysts, database designers, and lead developers. Activities performed by developers at Minerva were inclusive of coding, planning, code review, mock-up, preparation for workshop, bug fix, park until technical fix, testing and designing cases
Lead Developer	A developer could also be assigned the role of lead developer, placing him in charge of a particular project with the additional responsibility of architectural design, writing code, testing, and estimation. At the onset of the project, the lead developer is usually paired with a *coding buddy* essentially forming initial teams of two.

The organogram below illustrates the employees at the Minerva:

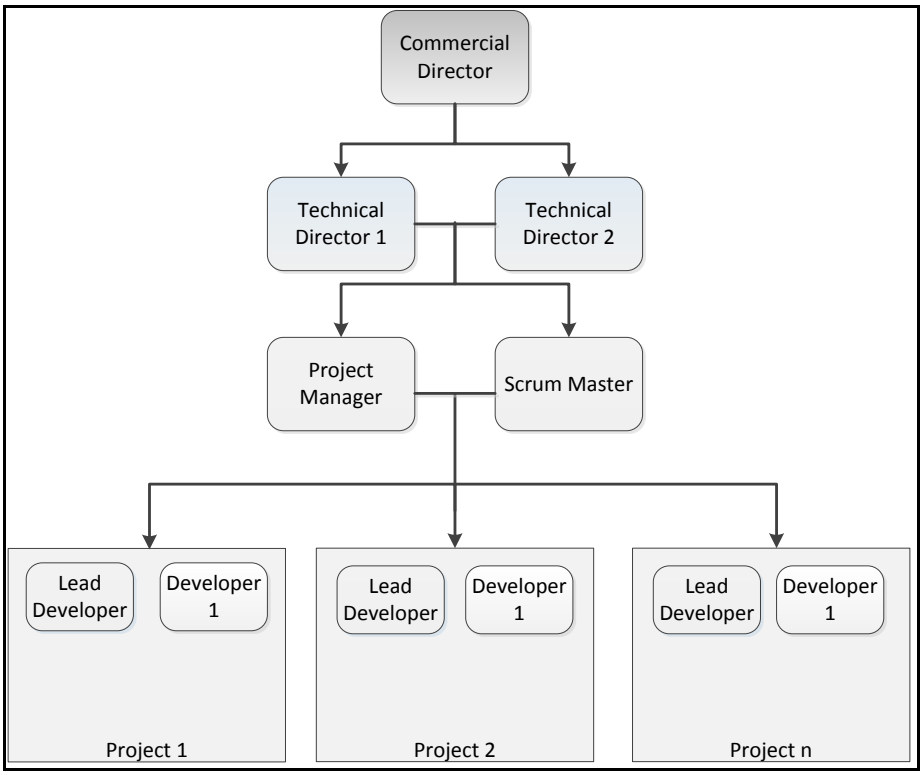


Figure 13: Minerva employee organogram.
Source: Researcher's own construction.

In both case sites, the entire workforce was observed, but not all were interviewed. For Athena, the CEO, Operations Director, Technical Director, Scrum Master, and six developers were interviewed. Of the two developers who were not interviewed, one was away for two weeks and the second did not concede to an interview on the basis of *shyness*. While at Minerva, the Commercial Director, one Technical Director, the Scrum Master, and eight developers were interviewed.

The next subsections turn attention towards the fieldwork conducted at each case study site.

5.5 Primary data collection techniques

Observation and interviews formed the major conduits for exploring program management practices. These are described individually in more detail in the following subsections.

5.5.1 Primary data collection technique: Observation

The details of the observation technique and guidelines for ethnography were elaborated on in Chapter Three. At Athena, the researcher was installed in the open plan office which afforded a good vantage point to overhear conversations and occasionally engage in these

conversations. As commented on by a developer at Athena, watching developers write code was not considered conducive nor productive towards data collection, but the closeness allowed for interesting *chatter* to be noted. For example, there was an instance where the Scrum Master was discussing experienced technical difficulties with a developer. A second developer assigned to a different, concurrent project joined the conversation in an ad-hoc manner offering his insights since his current project experienced similar problems. It was not long before three other developers were embroiled in this conversation. In the end the Scrum Master established a notion of a certain commonality in technical difficulty and this in turn influenced his cross project planning. This was easily observed and documented due to the location of the researcher. Observation was also carried out when attending meetings such as daily stand-up meetings and workshops.

At Minerva, the researcher shared an office with the Scrum Master. This decision was taken by the CEO who understood the context of this study and felt that this location was most beneficial for the researcher. Observation allowed for documentation and conversation with the Scrum Master as he went about practicing program management. Misunderstandings or gaps were quickly elaborated on by him before being documented. At both sites, the researcher observed activities that often led to resolutions and formulation of resultant action plans during meetings and workshops.

5.5.2 Primary data collection technique: Interviews

The procedure and guidelines for interviews was presented earlier in Chapter Three. Interviews were held during the last week of the field work, at each case site. For both cases, all levels of developers, Scrum Master, and CEO were interviewed on a one-on-one basis. Interviews lasted 45 minutes for participants and up to 80 minutes for Scrum Masters, senior personnel and the CEO. For both case studies, participants were willing and enthusiastic to devote time to interviews due to their interest in the research. The first two respondents at Athena served as pilot interviews but no significant changes were required for the interview procedure. Activities or processes that still remained mysterious after the observation period were clarified during interviews. All interviews were recorded and the participant was first asked permission. No participant who had agreed to be interviewed objected to this. Interview recordings and notes were transcribed immediately after the interview or later the same night. Not surprisingly, most participants preferred not to have the interviews videoed and this avenue was not pursued further. Interview questions focused on program

management practice activities with themes being identified after observation providing a loose structure. Knowledge areas and practices identified during the literature review were also used to supplement and formulate interview questions.

5.6 Secondary data collection

For collection of secondary data, the researcher attended daily stand-up meetings, innovation and sprint retrospective workshops, joined informal discussions, attended sprint grooming sessions, sprint release planning sessions, viewed the daily updated project charts, viewed project management software artifacts, and was provided with feedback from client sprint reviews. These are described in more detail in the following subsections.

5.6.1 Daily stand-up meetings

Daily stand-up meetings at Athena revolved around a Kanban board. A Kanban board is a type of visual aid borrowed from Lean production guidelines of visualizing work. It provided a *finger on the pulse* type mechanism for each project. Kanban boards visualize the software development progress, help identify *roadblocks* or slow moving tasks, and provide an indication of work remaining. Project burndown and velocity charts were presented by the Scrum Master during daily stand-up meetings. These charts were produced by the project management support tool named Pivotal Tracker, a commercial off-the-shelf software package. At Athena a separate daily stand-up meeting was conducted for each project. Some developers had to attend more than one daily stand-up meeting due to their involvement in multiple projects. Daily stand-ups were held mid-morning or early afternoon at the same time each day for the duration of the project.

At Minerva Kanban boards were discontinued. The following narrative motivates this course of action:

> "They create a production line mentality with too many developers focusing on maintaining the Kanban instead of other traits like learning new technologies" min_dev2_obs_pg7:16

For Minerva the daily stand-up meetings revolved around a spreadsheet prepared by the Scrum Master. The data projected by this spreadsheet originated mainly from the project management support tool called FogBugz, also available as commercial off-the-shelf software. This spreadsheet provided a holistic view of progress on all active projects, affording the entire staff a means of deliberation. For Minerva, the entire staff attended one daily stand-up meeting, and those staff that were out of office (for example, on a client visit

or any other activity other than leave) participated in the daily stand-up meeting via group conference call. During stand-up meetings, developers wrote down and read out progress using their own materials such as iPads and handwritten notes. Developers reported on work done, what they were waiting for (for example, waiting for completion of a code review or for testing results), the work load for the day, and challenges.

For both Athena and Minerva, daily stand-up meetings did not usually go on for more than 20 minutes and were conducted religiously every day.

5.6.2 Innovation workshops and developer's meetings

At Athena innovation workshops were held on average once per sprint. The researcher attended these innovation workshops which were two hours in duration on average. Developers took turns to choose a relevant topic for discussion and to facilitate the workshop. Topics of discussion ranged from new technological developments to improving the team experience. Some innovation workshops at Athena focused on mental development exercises in the form of friendly contests and team building.

Weekly developer meetings were held at Minerva which the researcher attended. The primary intention of these meetings was knowledge sharing among the development staff. Similar to Athena, developers facilitated the meeting on a rotational basis. This was a purely technical session where issues and experienced problems were confessed and deliberated over. Topics included programming techniques, new program components, and useful websites, blogs and online chats. Any artifacts that featured in the workshops were catalogued and stored in Microsoft OneNote. Featured artifacts typically included scripts, links to forums, links to blogs, code classes, code snippets, and documentation from research and innovation. OneNote was the central artifact of the developer meeting.

Innovation workshops and technical meetings were instilled primarily for knowledge dissemination and sharing across projects and personnel; hence, they are viewed as part of program management. Finally, attendance and participation in these workshops reduced alienation of the researcher.

5.6.3 Sprint retrospective

The researcher attended sprint retrospectives which took place at the end of each sprint at both Athena and Minerva. Sprint retrospectives were mechanisms that supported reflective learning and aimed to satisfy the agile principle of continuous improvement of the

development process. Developers, Product Owners, and Scrum Masters attended. The formats of the retrospectives varied between Athena and Minerva; but the objective was the same as described by a senior personnel member at Athena:

> "The primary feature of Lean development is flow. We should try to maintain a good flow by unblocking any blockages. We should avoid a rush on Friday. Instead, we should have consistent flow throughout" ath_od_obs_pg15:7.

Recall that a sprint retrospective aims to improve the team's efficiency in software development through reflection of practice and devising plans for improvement. This reflective practice usually adopted the form of posing questions along the lines of: *What went wrong? What went right?* and *How to improve on what went wrong*? Good practices were also highlighted as the team wished to preserve them. For Minerva, this generic procedure is followed for sprint retrospectives. At Athena, a slight variation on this generic practice was noticed. Here the team professed their opinions of the previous sprint along a matrix of categories such as love-loath, like-dislike, want-don't want, and learnt-forgot.

In both cases, references to management were made during the retrospective. That said, mechanisms for conveying these points to the management team, and documentation of these points, were not elaborate in both cases. A developer had this to say:

> "We have a total antithesis to documentation over here" ath_dev3_obs_pg12:2.

How then were these lessons learnt and how were experiences from the project space spread across the SMME? This question will be explored in Chapter Six.

5.6.4 Product backlog planning sessions

Recall that a product backlog is a prioritized list of functional requirements for a single software project. It signifies what the software product must do. At both case sites, functional requirements were captured in user story format. The product backlog was created after initial requirements and business analysis, which was performed in concert with the client. A product backlog planning session was held shortly after initial requirements and business analysis which the researcher sometimes attended. For Athena, this process involved the Operations Director and one or two developers assigned to a project, while at Minerva this responsibility fell onto the lead developers. (The concept of lead developer will be discussed in more detail later). For Athena, user stories were recorded on post-its and pasted on a large wall chart, providing a visual representation of the product backlog. The product backlog stories were then captured in Pivotal Tracker. For Minerva, the product backlog user stories

were planned on paper first then captured on FogBugz. Athena had adopted a system of assigning a number to indicate effort and priority for each user story. Minerva instead opted for t-shirt sizes to indicate effort, as is indicated by the following comment:

> "This was done in an attempt to do away with numbers. The management team found that numbers were being assigned based on the time in hours needed to complete the story" min_dev3_obs_pg3:5.

For both cases, the product backlog evolved over time to reflect the most recent understanding of project functional requirements.

5.6.5 Sprint release planning and grooming sessions

Sprint release planning started once the product backlog was agreed upon for a project. The aim of sprint release planning was to convert each product backlog user story into a collection of smaller, more technically inclined sprint user stories or tasks, and then situate each within a sprint for the project. The end result was a release plan. At both Athena and Minerva the developers decided on the required effort for completing each sprint user story, while its priority was derived from the product backlog. The Scrum Master or any of the other management level employees, were not usually involved in this process unless they were developers on the project. At Athena, the release plan was visualized on the Kanban board and captured in Pivotal Tracker. At Minerva, the release plan was captured directly into the configuration management software and no physical visual aids were employed. The grooming session involved adjusting the effort and priorities assigned to each user story in the release plan as the project progressed.

5.6.6 Feedback from sprint reviews

For Athena and Minerva sprint reviews occurred at the end of each sprint, which roughly equated to a frequency of once every four to five weeks. Software was demonstrated to the client and feedback was elicited. Feedback appeared in the form of criticisms stemming from demonstrated software, advice on enhancements, and additional functionality requests. The reluctance to extend an invitation to the researcher was an indication of the clinical importance of maintaining good client relationships. The researcher respectfully did not pursue this avenue. However, discussions held with the developers who participated in the sprint reviews revealed that the outcomes from these meetings interface with program management practices by influencing planning, resource allocation, and budget.

5.6.7 Informal communications

Athena had an open plan office in accordance with agile guidelines. The entire workforce, except for the CEO, was seated in the same room. This enabled and encouraged informal communication and transference of osmotic sounds (Cockburn, 2007); that is, information that is shared by listening in on the conversation of others. Minerva had a variation of an open office whereby just the developers sat in an open plan office. Many of the senior developers sat two in an office, with the joint CEOs having an office each. Minerva maintained quiet hours in the morning and early afternoon. During these designated hours no conversations, whether inter-team or telephonic, took place in the office area. Instead, developers were encouraged to converse in the boardrooms or outside. The researcher was completely unaware of this rule until the day he had a lengthy, personal telephone conversation during one of these designated times. The next day during a short, informal interview a senior employee eloquently mentioned in passing the quiet hours etiquette. Great care was taken by him not to offend the researcher.

This specification of open office falls within the ambit of program management because it is the software development equivalent to the task of organizing the factory floor in a manufacturing environment. Lean principles present certain guidelines for improved efficiency which can be applied to organizing a factory floor or the production process. For agile, the open plan office is a guideline in lieu of improved informal communication and project awareness.

5.6.8 Software support tools

At the time of fieldwork, Athena used Pivotal Tracker, a commercial off-the-shelf software support tool.. A visual readout displayed the company's velocity; that is, the velocity across all projects. Velocity and burndown charts were produced by Pivotal Tracker. The Scrum Master used these charts to highlight good or problematic areas in current projects. A slightly different system was implemented at Minerva, in the form of custom developed software that interfaced with FogBugz. The Scrum Master used the information produced from these combined systems to generate a spreadsheet representation of progress which was used to drive the daily stand-up meetings. In both cases reporting on individual project progress was influenced by the project management software and performance charts.

5.7 Summary of chapter

This chapter introduced the case study sites via biographical and background information as well as descriptions of personnel roles and responsibilities. It then presented an overview of the fieldwork by describing the primary and secondary data collection techniques in the context of the case study sites.

Both case study sites were of similar size in terms of number of employees and appetite for simultaneous software projects. Although the employee role names were different at each site, the responsibilities were similar. The organogram of employees was similar with the only major difference being at the lowest hierarchical level. Minerva utilized a two person development team while Athena had up to six developers in a team.

Data collection techniques were primarily observation and interviews. Observation at both sites was aided by permitting the researcher free passage to the open plan offices. Open plan office format greatly enhanced the observation technique. Interviews were conducted at both sites towards the end of the fieldwork. For both cases it allowed for resolution of queries and lingering questions, and more importantly, revealed information that could not be netted by observation alone.

Secondary techniques included attending Scrum rituals and ceremonies. At both sites the researcher was fortunate to have been given permission to attend all meetings except for the demonstrations of software to clients (sprint reviews). Sprint reviews were important intimate milestones for building client-SMME trust and the researcher did not pursue an invitation. Although participation in these meetings was easily achievable, this was guarded against to avoid bias and unnecessary influence on the outcomes.

Chapter Six provides an analysis of the data collected at the case study sites.

CHAPTER 6: DATA ANALYSIS AND PRESENTATION

6.1 Introduction

This chapter presents an analysis of the data collected at the two case study sites. Activity Theory (AT) is used as the data analysis lens to structure the analysis and provide a view of the interplay between program management practices, SMME management characteristics, and Scrum. The program management knowledge areas and associated practices presented in this chapter were informed by both the literature review and the fieldwork. Initially the literature review (Chapter Two) provided insights into program management knowledge areas and also afforded a cursory view of key associated practices. These served as initial indicators during the early stages of data analysis. However, additional knowledge areas and practices were unearthed during the course of the fieldwork. This thesis does not pretend to produce a comprehensive documentary of program management knowledge areas and practices in the context of Scrum and SMMEs, and acknowledges that some practices could have been inevitably overlooked despite the long period of fieldwork.

AT proved an appropriate lens to describe program management practices of both case study SMMEs because it succeeded in exposing a diverse range of socio-technological interaction, such as ICT technologies, people, rules and methodology, culture and social norms, and company strategy. Once exposed, these interfaces were then systematically unravelled through more involved fieldwork techniques such as interviews, and minimal researcher participation. Owing to the complexity of observed program management practices, each knowledge area and associated practices is described as an activity system in its own right.

This chapter is structured as follows: The first section provides a recap of relevant concepts of AT, and offers an analysis of program management practices at Athena, followed by the analysis of Minerva. The chapter ends with a summary.

6.2 Representation of program management knowledge areas as Activity Theory sub-activity systems

What were the eventual program management knowledge areas? After the literature survey (subsection 2.5.6), and upon completion of the fieldwork, program management knowledge areas were distilled by the researcher into the set below:

a. Staff allocation
b. Coordinating activities
c. Cost estimation
d. Human resource management
e. Infrastructure design
f. Program monitoring and reporting
g. Planning
h. Client liaison
i. Schedule risk identification and mitigation
j. Alignment of practices with business strategies
k. Learning practices

This list was not designed to represent a concise set of knowledge areas. Instead, the researcher's best effort resulted in this list of knowledge areas, and these are expanded in this chapter.

For the purpose of recapping the epistemological concepts of AT (subsection 3.8.6), the diagram below depicts a simulated representation of program management as an activity system:

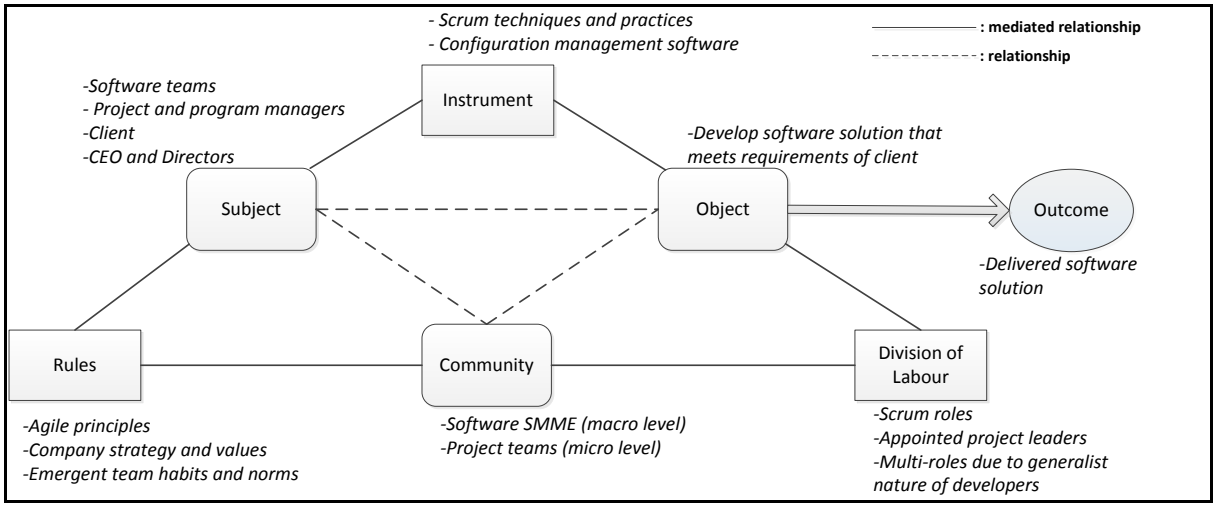


Figure 14: Simulated activity system for program management.
Source: Researcher's own construction.

The *subject* component includes individuals and groups of individuals participating in this activity system. *Object* is the target or objective which represents the aim and purposeful nature of the activity system; that is, the reason for this activity system's existence. *Community* situates the activity system in the context of the social and cultural environment. These components exist as relationships and not in isolation. Each of these relationships are analyzed by taking cognisance of mediators that serve as inhibitors or promoters of relationships. Examples might include *instruments* (physical or conceptual tools), *rules*

(explicit, in the form policies or implicit, in the form of cultural norms), and *division of labour* (responsibilities and variations in job roles).

In an attempt to more concisely expand on the object of effective program management (illustrated in Figure 14 above), sub-activity systems were devised. This represents a slight departure from the typical application of AT and is considered innovative in its approach. Each of the program management knowledge areas is represented as a sub-activity system. These then culminate to form the overall object for that particular program management knowledge area. Each sub-activity system is further decomposed along three AT-derived dimensions, namely:

a. *sub—inst—obj*: The relationship between personnel (subjects) and the specific program management knowledge area (object), mediated by techniques, tools, and project artifacts (instruments).
b. *sub—rule—com*: The relationship between the personnel (subjects) and the software SMME (community), mediated by guiding principles and core values (rules).
c. *com—div of labour—obj*: The relationship between the software SMME (community) and the specific program management knowledge area (object), mediated by different responsibilities of personnel (division of labour).

For readability purposes, these relationships are denoted by sub—inst—obj, sub—rule—com, and com—div of labour—obj (with the mediator, be it the instruments, rules or division of labour, located in the middle). Although the epistemology of AT suggests that each relationship is mediated, it must be pointed out that some mediators could not always be detected or were sometimes absent altogether. Further to this, it must be stressed that the relationship rendered in these sections represents a point of view of the researcher. It is quite possible that another researcher conducting the same analysis using the same lens and data set might portray a different permutation of relationships from that presented here. The analysis begins with Athena, the first of the two case study SMMEs.

6.3 Analysis of Athena case study data

Athena was the first case study site. Program management knowledge areas and practices identified during the literature review provided early clues. However, the actual observed practices did not always strictly conform to those identified a-priori and as prophesized; some new knowledge areas and practices emerged. Eventually, observed program management practices at Athena were categorised by knowledge areas; these include staff allocation, coordinating activities, costing (cost estimation), human resources management,

infrastructure design, progress monitoring and reporting, planning, client liaison, risk identification and mitigation, business strategy alignment, and learning.

The next subsections provide descriptions of each of these knowledge areas and practices at Athena, beginning with staff allocation. In each case, the notation relating to relationships as outlined above is used. (Whilst this may seem jarring to the reader at first, putting this information into a table would have disrupted flow of meaning, given the need to add supporting references from the fieldwork to underscore what is noted).

6.3.1 Staff allocation

The staff allocation knowledge area aimed to ensure adequate and competent personnel were allocated to the various concurrent projects. Allocation of developers across projects was highly fluid. Although automated project management support tools in the form of Pivotal Tracker aided in this practice, the small workforce also helped with maintaining tacit awareness of current assignments.

Sub—inst—obj: Pivotal Tracker is the project management software (instrument) that provided an indication to the CEO and Scrum Master (subject) of the current work assignments (object). The Kanban board (instrument) indicated a project's progress per sprint which in turn influenced staff allocation (object) performed by CEO and Scrum Master (subject).

Sub—rule—com: Developers (subjects) were elected for a project based firstly on their workloads (rule); that is, a developer with a lower or no current workloads indicated availability within the developer workforce (community). A developer was assigned to one project but a developer may have been working/helping on other projects simultaneously. Secondly, experience and technological skill (rule) were deciding factors when assigning developers to a project. The Scrum Master explains:

> "Two things. One availability. Couple of devs were pretty well tied up in existing projects. So in order not to rock the boat on those existing projects, we pulled in other devs who were doing more piecemeal stuff. So that's on the one side. On the other side, skills. For example the development side [dev1[2]] and [dev5] had been working on small projects doing support, that kind of thing, hence it made sense to pull them in. On the UI side we pulled in all the resources we had. We didn't have a choice. [dev3] was working 3 days a week as UI designer and [dev6] as a graphics designer. So, that's all the resources we have in that area so we put them all in" (ath_int_sm_pg10:3).

The technical director supported this description of practice:

[2] [dev*N*] denotes a developer. Names have been omitted to preserve anonymity.

> "There's a lot of thought that goes into that. One is availability within the company. But also, who has the skills that suit the project, who needs the opportunity. There's a lot of factors that go into it. Because the project required a lot more UI we brought in [dev3] more heavily" (ath_int_tdir_pg6:4).

The remark above reinforced the trait of SMMEs having limited human resources. Dev3 in the narrative above was a freelance UI developer and was sometimes present for up to 4 days a week.

Com—div of labour—obj: Developers employed at Athena (community) could have assumed the role of programmer, tester, analyst or user interface designer (division of labour) when being assigned to a project (object). However, for the most part, developers assumed roles within their area of expertise. Senior developers played the additional role of mentor (division of labour) when paired with new or less experienced developers.

> "Encourage them to pair as much as possible so seniors can teach the people that are less experienced" (ath_int_tdir_pg5:4).

Further proof in the instance of a newly hired developer:

> "He has the potential to work well and he just needs to follow one of us around more closely. We can get him up to speed eventually" (ath_obs_pg8:3).

Despite this, during the course of the fieldwork at Athena this newly hired developer resigned.

> The CEO informs a project team that a meeting was held with the new developer and that he is very concerned with two aspects. First, he feels overwhelmed by the agile practice, techniques, self-management etc. stating that he feels that he is not good enough; and secondly, he does not want to disappoint his team. CEO then points out that she attempted to improve his morale and confidence. (ath_obs_pg8:4).

It emerged that within an SMME like Athena, frequent hiring and firing of personnel can sometimes escalate to a costly exercise. The time and effort dedicated towards nurturing and allowing a new employee to build experience requires expenditure of limited resources, making staff turnover a costly event.

Within this knowledge area the practices of allocating staff to projects depended on their skill set, prior project experience with the client, and availability. Cross project work at Athena was common. The extract above pertaining to the resignation of a developer within a short period of being hired, demonstrated the need for a certain aptitude that was required to work in a highly dynamic project environment. In some cases less experienced staff were allocated to projects with a senior developer in an attempt to alleviate some of this pressure. This practice also facilitated with skills development.

6.3.2 Coordinating activities

Detailed development work taking place within each of the concurrent projects required constant coordination. Coordination activities mainly entailed planning of tasks, task allocation, and progress monitoring for each concurrent project. Recall that a unit of work was represented by a user story and that these user stories each represented a functional aspect of the software system. Allocation of user stories was resolved by the team of developers.

Sub—inst—obj: Coordination of work (object) within the project was done by the team of developers (subjects) on a daily basis during the daily stand-up meeting (instrument). A Kanban board (instrument) and Pivotal Tracker were the primary tools which supported developers (subject) in coordinating their activities (object).

> "Visual management, I think, is critical. Be it the graphs that I share with the guys, be it the burndowns, the stories moving across the Kanban, and code coverage graphs, those things are critical to communicate to the team, this is where we are guys, this is something that needs to be addressed. But on top of that the team needs to know how to interpret that as well. Because just putting it up every day doesn't mean anything if you are not drawing attention to it. You know what guys, this is pretty bad and this needs to be looked at" (ath_int_sm_pg12:2).

These tools (instrument) provided the team (subject), Scrum Master (subject), and any other personnel (subject) with an overall indication of project status (object) using user stories as a measurement metric. For Athena these statuses included *completed*, *work in progress*, *in testing*, and *done*. This information was easily visible and sustained project awareness was vital for coordinating activities within the individual projects.

Sub—rule—com: Developers (subjects) assigned to a project were responsible for coordinating activities (rule) within that project (community). The Scrum Master or any of the senior personnel (subjects), including the CEO (subject), could force a change (rule); nonetheless, they mostly played the role of *devil's advocate* (rule) and assisted the team in difficult situations (rule) during the course of the project (community). The following quote explains this devil's advocate role:

> "Management influence it in terms of looking at where the team is failing and trying to suggest ways in which they can do better and trying to ask the right questions in terms of why are these things failing. So, to not just come out and say you have to do this and the team has to do this. Unless of course we are in absolute crisis and no one is picking it up. So you won't have [Operations Director] coming in and saying you guys need to stop doing it this way and go here. Ideally he just wants to be watching and looking to see, ok I can see it failing here and subtly hint and then, obviously, if it continues then he will decrease the subtlety. Obviously if the project gets to a point where it is completely failing and it is completely in crisis, then he's got a job and he's got to step in and do something about it" (ath_int_dev1_pg17:2).

Work was not assigned to a developer (subject) in a top-down hierarchical manner; instead, a developer chose his/her own tasks (rule) for the next day or two for the project (community).

> "It's managed by culture more than a direct managerial directive as to how things work. So management's way of managing that, is to create a culture in which it can exist as opposed to trying to implement, to restrict solutions, and a strict structure to it" (ath_int_dev3_pg24:2).

> "Independence is quite important. There aren't a lot of n*anning* happening and there's trust that you'll do your work. One has to be able to do their work without constantly needing attention" (ath_int_dev4_pg25:1).

Developers are "trusted" to be responsible enough to allocate work to themselves, but at the same time:

> "Are you confident with the work/tasks for today?" (SM).
>
> Scrum Master states that team needs to achieve 8 points today. Developers feel that this is tough. One of the developers points out:
>
> "We need more accuracy in terms of the sprint planning tasks for the week and the prioritization of the stories" (dev2).
>
> "That's fine, we can look at prioritization but we must also ensure that stories come through. We should concentrate on delivering value adding components to improve our poor flow rate and increase points" (SM).
>
> (ath_obs_pg6:6).

Although developers select their own tasks, this activity was not completely devoid of persuasion from management.

Com—div of labour—obj: Within the project (community) all developers (division of labour) played a role by coordinating their own activities (object). This highlighted the inter-team, self-organizing nature, seeing as senior personnel rarely assigned tasks. Developers had areas of specialities; such as testing, UX designer, or C# programmer (division of labour), and usually selected tasks (object) accordingly within currently assigned projects (object).

At the time of this fieldwork, one of the projects was in the proverbial red. Interestingly, during this period the CEO and senior personnel (division of labour) were persistently trying to instill within the project team a sense of ownership and responsibility (objective) for the overall well-being of Athena (community). Hence, approaches such as the team's self-coordination of project work can be seen as a potential support mechanism for this ethos.

> "But a very big part of what [CEO], [Operations Director], and [Technical Director] is [to] have the team work it out, for them to figure it out. To start asking the right questions to start doing the problem solving and part of that is stepping out and almost letting the team fail. Because you never get the sense of *I can fail at this* because if someone is always jumping in to rescue you, you tend

> to think it's ok, it's fine, because at some point this person is going to jump in and rescue it" (ath_int_dev1_pg17:2).

Coordination of tasks was done by the developers with help from the Scrum Master. Higher order personnel did not influence the allocation of tasks within the team. Team members seemed to balance choice of tasks within their area of expertise against venturing into new areas when the project was not under duress. A noteworthy observation was the fact that the senior personnel upheld the *right of passage* mentality for newer or less experienced staff. They had to learn from their mistakes and in the process developed a strong sense of ownership of their actions, with a consequent positive ripple effect on Athena.

6.3.3 Cost estimation

This knowledge area included activities of cost estimation and cost-related reporting of the various projects underway. There were two broad categories of projects at Athena. The first was a fixed schedule and budget, while the second was billing per hour for work done and was more commonly referred to as time- and materials-based projects. There was no billing per hour projects underway at Athena at the time of fieldwork. Therefore, the data analysis primarily related to fixed schedule and budget type projects.

Sub—inst—obj: At the outset of a project, the Product Owner (subject) created the product backlog (instrument) which contained user stories for the entire software system (object). The developers assigned to that project (subjects) performed planning by using the product backlog (instrument) which resulted in a release plan (object). User story estimates (object) were completed by the developers (subject) and captured on Pivotal Tracker (instrument).

The Scrum Master (subject) relied on progress reports; for instance, burndown charts (instrument) which were generated by Pivotal Tracker (instrument) to determine the amount of work that remained in a particular project (object). A spreadsheet (instrument) was maintained that recorded this data from different projects (object) and was updated on a daily or sometimes twice daily basis by the Scrum Master (subject). This helped provide a holistic view of work remaining against the allocated budget for each project. This multi-project information was reported to the CEO and Directors during weekly meetings.

Sub—rule—com: Initial cost estimation for a project (rule) was done by both CEO and Directors (subject) of Athena (community). This incorporated overall budget, profit markup, and profit/loss parameters. The developers (subject) performed estimation for each user story (rule) within their assigned project (community). The Scrum Master then assigned a cost for

each user story abiding to margins of the initial cost estimation. The Scrum Master (subject) reported on financial status of each project (rule). There was an interface between the CEO (subject) and Directors who performed the initial estimation (rule), the team (subject) who performed project level estimation (rule) for work that needed to be done, and the Scrum Master (subject) who reported on progress across projects (rule). An extract from the Scrum Master's interview may provide further clarity on this interface:

> "For [project A] we looked at the spec; well when I say spec I mean requirements because it was very open, there wasn't much of a requirement. We figured out what they want, they showed us their existing system and [Operations Director] and [Technical Director] went through it and decided on how much it will be approximately. It is effectively sit down and go, well it's going to take that developer so long and add up the hours, multiple by rate, and add 10% or whatever the margin might be. So it's an estimate but you get better and better at it when you've been in software for a long time" (ath_int_sm_pg10:5).

The Technical Director also highlighted parameters that influence cost estimation:

> "From a company perspective we try to set rates that we think we should be earning. But we've got historical agreements with customers that we've done a few years ago and we've just been increasing those rates and trying to get them online with what we think our company rates should be. So there is a sort of overall pricing policy but it is not really formal. We don't say alright, this is the price for you because it really depends on the type of work, if its long-term work you get a cheaper price etc. But for a senior dev we do have a price point, a range for our developers. When I do a quote, I'll keep that in mind, but also keeping the customer in mind and the type of work. There is also room for negotiation" (ath_int_tdir_pg7:3).

This "not really formal" approach to cost estimation is amplified in the following narrative:

> "All those roles don't have to be played by separate people. So essentially we would look at the project and estimate the hours per role rather than per person. So we can say that this project requires 350 hours of BA work, 750 hours of development work, 800 hours of UI work, and 500 hours of UX work. This would be the base cost with some profit built in there. This is pure billing per hour and this would be the perfect agile situation. But most times we have this semi-agile situation where the budget is actually fixed, the time scope is more or less fixed, but what actually gets done gets chopped and changed. So we have to fit in a mark-up to cater for the risk involved in changing of scope and changing of functionality. That's pretty much how I think costing is done" (ath_int_dev1_pg15:1).

It appeared that cost estimation is still very much a practice with strong tacit undertones. The following comment provides further clues as to why cost estimation exists in this format:

> "Projects don't get costed at the moment. It's certainly something that we want to do but it is a complex thing. So we've got these rates that are based on a number of things. They are based on the cost of the developer plus the overhead" (ath_int_tdir_pg7:5),

and:

> "Fixed pricing in an agile world is very difficult. Because it requires accurate, accurate, accurate prediction of all the factors that are going to happen and the whole principle of agile development is that we don't know. So I think that we need a flexible, fixed priced estimate initially and the

client understands that that can be adjusted depending on how the project changes down the line" (ath_int_dev3_pg22:1).

For reporting purposes, indicators were always visible (rule) to personnel and management (subjects) via reports and graphs generated by Pivotal Tracker and by the Scrum Master per project (community). Any additional functionality request from a client triggered negotiation (rule) between client and senior personnel (subjects) over delivery date and sometimes escalated to additional cost. However, escalated costs were not observed for any of the projects during the fieldwork. Instead the client request was honoured at the expense of the SMME or in conjunction with an agreed upon later delivery date. The following extract from observation notes highlight this notion:

"We are willing to lose money on this project just to maintain good client relationship" (CEO)

"As it is I hear that requirements have changed in UK" (Dev3)

"That's the price to pay to build a relationship" (CEO)

This sentiment has to be considered alongside the fact that the team being referrred to is new; that is, its members are new and this is the first time the team is working in an agile, self-management based team. Also, there are concurrent projects taking place; hence, Athena's management and CEO may be willing to lose a little on this project with the hindsight that the team is developing and learning the ropes. To some degree, this contravenes the SMME characteristic of limited tolerance for loss; however, the proverb *penny wise pound foolish* was recited by the Scrum Master when quizzed on this.

The group then speaks of pricing:

"Coming from a fixed price project, optimizing pricing is difficult in agile." (CEO)

"Not exactly fixed budget, but fixed on salary rates etc." (Dev3)

(Ath_obs_pg32:2)

Com—div of labour—obj: Developers of the project (community) set estimates for user stories (object) which represented an addition to their main responsibility of designing and writing code (division of labour). Their user story estimations were used to generate the projected duration and delivery milestones for the user stories which in turn led to the calculation of cost of the project.

The Operations Director spent a little time elaborating on the reasoning behind having developers instead of management perform the estimation, which was in contrast to the usual modus operandi of software engineering.

> The Operations Director states that it is better to assign points at a lower level of breakdown. His theory is described by using the analogy of flipping a coin. When flipping a coin there is a 50/50 accuracy. If we flip four times, then 25% accuracy, 8 times then we start getting a bell graph type of accuracy. So the more 3rd level stories are used for estimation will lead to a more accurate estimate at the 1st high level story. 3rd level is a lot more detailed and used for daily work breakdown stucture which the developers then code and the UI guys use to design. (ath_obs_pg38:1).

The Scrum Master compiled and reported (division of labour) on expenditure for each of the projects (object), mainly during the daily stand-up meeting for each project (community) and every two weeks during a management meeting.

The Scrum Master was mostly in charge of monitoring conformance to budget and reporting. Reporting of budget was real-time and changes were noted at least on a daily basis. This negated the need for a *heavy* upfront cost estimation plan. An interesting interface was uncovered within this knowledge area. The CEO and Directors established the initial cost for a software project and through negotiation with the client, settled on a delivery date. The developers performed the estimation and planned their work according to margins dictated by that delivery date. The activities associated with cost reporting often rested upon the Scrum Master; although the initial budget allocation for each project was determined by the CEO with assistance from the two Directors. In addition to the traditional software engineering approach where cost estimation usually occurs early in the project, at Athena estimations were also done in real-time with updated indicators available throughout the day. Furthermore, estimates were done by both senior personnel and the developers assigned to a project, and not solely by the higher level personnel. Given the dynamic nature of software projects and the agile principle of embracing change, there was sometimes a mismatch and as a result management had to exert pressure on the team to work beyond the boundaries of *their* initial work plans. The alternative was to lose profit. Always, the client relationship was viewed as paramount. Narratives confirm the fluid and reactive nature of cost estimation practice highlighted by such statements as: “semi-agile” and “more-or-less”. Due to agile guidelines permitting leeway of functional requirements, initial cost estimation must have considerable margins to accommodate this emergent nature.

6.3.4 Human resources management

Human Resources Management is a vast academic field with many practices and knowledge areas. This thesis acknowledges this and does not pretend to provide concise and complete coverage of Human Resources Management. Instead, the area of interest was the c*raft* of juggling scarce human resources between projects. Interaction with personnel at Athena

revealed a shared belief that experienced and skilled developers are a generally limited commodity in the software industry, and that SMMEs often have to compete for this resoure with higher paying corporates. Athena often faced the challenges of limited human resources and simultaneous projects.

Sub—inst—obj: Pivotal Tracker and the Kanban board (instrument) provided the Scrum Master and developers (subjects) with an indication of where human resources were committed (object). These project management support tools, together with data from burndown graphs and feedback from daily stand-up meetings, provided indications of imminent and lurking danger in terms of project schedule. This awareness played a pivotal role in the decision-making process when developers were moved between projects. Pair programming (instrument) was another mechanism that supported movement of developers (subject) between projects (object).

Sub—rule—com: Project teams typically consisted of 4-7 developers (rule) that were hand-picked by the CEO and Directors (subjects) from the available developers (community). It was noted that these teams were dynamic (rule) in that both the number of developers and the delegated responsibilities within each project team (community) could have changed on-demand. However, management tried not to disrupt teams too much. On some occasions developers were moved between projects for a limited duration on instruction from either of the Directors or from the Scrum Master.

The CEO and Scrum Master (subjects) could manipulate project staff allocation by juggling or jostling (rule) for developers (community) for a project. However, this did not look like a seamless migration. When asked about the observed practice of adding additional developers to a lagging project, the Technical Director had this to say:

> "We do sometimes do that. But what we really want [is] a team to gel. So we want to choose the right-sized team in the beginning and then have them work together so they form a relationship and start gelling. And when you add another person, the team needs to almost reform and that takes time. So it is a little bit disruptive to do that. Like I say we've done it before and because of the way we work like we got sort of standard ways of architecting software. It's not that hard for someone to come in for a week and help because there's a standardized way of doing things…In the past we brought someone into a project for two weeks to knock one section out. Not because we were behind, but because the customer needed that extra thing. But also, you would have noticed that people hop out for a day and that is not too disruptive" (ath_int_tdir_pg6:5).

Developers were moved around at the expense of disrupting established team relationships. The development teams on some occasions were required to lobby for additional manpower. The following extract from observation notes highlight this notion:

> "The discussion then focuses on the new guy and the fact that he was pulled out of the project. This new guy is also new to the company and is learning the UI aspects. The lead dev complained to [Operations Director] that he should have been informed of his removal so he could have planned ahead. [Operations Director] then responds:
>
> "It is normal in [a] project environment for people to be removed etc. As a member of that project you need to fight for resources" (Operations Director)
>
> The team soon realizes that this company has many concurrent projects and that the company requires other work that needs to be done.
>
> A developer then asks:
>
> "Is this our responsibility?" (Dev1)
>
> "Well, management has a business to run" (Operations Director)
>
> (Ath_obs_pg18:4).

There were recorded instances of projects with dedicated developers who were not disturbed from current assignment. The Scrum Master explains:

> "We do try to keep the guys on their projects so they don't really have to work cross project most of the time. The only thing that we do on some projects is we have a rotating support roster for existing service level agreements (SLA) clients; if they do have any issues if any support is needed. Generally it rotates through all the devs. However for [projects] we've kept the devs off the support roster because we couldn't afford the time off" (ath_int_sm_pg11:5).

Athena employed freelance developers. These developers were sometimes asked to put in more working hours when projects began to lag behind schedule. On one project, contract developers did cause problems with planning of work and overall progression.

In other cases developers seemed to volunteer their spare time to help out lagging projects. It was not unusual for a developer with some free time (subject) to volunteer assistance (rule) in other projects (community) especially in those projects about to fall behind schedule.

> "That's a defining factor at [Athena], not really a concept of that's not my job. If something needs to be done the team climbs in and does it" (ath_int_sm_pg10:2).

Com—div of labour—obj: CEO, Directors, and the Scrum Master had the responsibility (division of labour) of moving or reassigning developers (object) to other projects at Athena (community). A developer may play different roles (division of labour) in assigned projects (community) to help achieve stability in that project (object).

At Athena, development teams were responsible for lobbying for additional manpower to bolster progress in their team. Furthermore, fieldwork revealed that project teams had to formulate their own resolutions when human resources were diverted from their projects. However, the Scrum Master and Directors tried to be the least disruptive to projects through

deliberations with team and expert judgement calls. Reasons for this included the ideology of allowing project teams to gel; that is, to establish healthy working relationships, learn one another's strengths and weaknesses, and to effectively plan around this understanding. However, trying not to be disruptive with a small compliment of developers was not without difficulties. A cultural observation was the willingness of developers to help out in their spare time. This was done completely voluntarily. It was a mixture of personality traits and close relationships due to *small shop* characteristics that fortify this culture.

6.3.5 Infrastructure design

Infrastructure design encompassed activities that ensured the preservation of the physical environment, such that it remained conducive to the nature of work being carried out at Athena. Although this activity outwardly appeared to be the preserve of CEO and Operations Director, comprehensive examination was not possible in isolation from the *men and women in the trenches*. Developers had an influence in infrastructure design. Hence, infrastructure design was placed within the realm of program management. Infrastructure included the layout of office equipment such as desks, computing hardware, visual aids such as product backlogs and user story boards, and electronic support tools such project management support software and e-mail. As noted before, there was one large open plan office at Athena that housed all developers, Technical and Operations Directors, and the Scrum Master. Only the CEO had a separate office. But at the time of writing, office space was at a premium.

Sub—instrument—object: There weren't any observed instruments in support of this practice.

Sub—rule—com: Open office plan (rule) allowed for easy informal communications due to close proximity of developers (subjects) working on different aspects of the same project (community). Visual aids of all projects (community) were situated such that visibility (rule) was possible to the entire workforce, including the CEO and Directors (subjects).

ICT and software infrastructure was purchased or built, based on project needs, advancements in technology and on the findings of developers (subject) who appeared to often perform background research and experimentation into these aspects (rule). This was more often than not triggered by project (community) needs. Also, being a SMME (community), efficiency in operation was a constant quest (rule) and software technology which had the potential to assist in this regard was often tested by developers and Technical Director (subjects).

Com—div of labour—obj: Developers, CEO, and Directors (community) each played an influencing role in setting up the various electronic, software, and physical components (division of labour) that constituted infrastructure design (object). CEO and Directors were responsible for the initial layout of physical infrastructure. This layout supported the agile principles of informal communication and information sharing and did not change for the duration of the fieldwork. Developers made suggestions in an attempt to streamline their daily tasks; such as the suggestion to purchase multi-coloured post-its for the Kanban board.

At Athena infrastructure and its layout was very much in line with conventional agile practices and recommendations. The researcher was situated in the open plan office and observation revealed a mixture of light-hearted camaraderie intertwined with periods of intense silence and concentration. Informal communication, osmotic noise, and help literally at the *snap of a finger,* were observed on a daily basis.

6.3.6 Progress monitoring and reporting

Under this knowledge area observed practices focused on the monitoring and progress reporting of concurrent projects. These reports provided a *finger on the pulse* for each project which in turn facilitated the higher-order decision-making performed by the CEO, Scrum Master, and Directors. Information from the various projects flowed outwards via rituals and supporting project artifacts; such as daily stand-up meetings, visual aids, and project management software. This information was mostly collated by the Scrum Master before being summarily reported.

Sub—inst—obj: The Scrum Master (subject) was the primary actor responsible for progress monitoring and reporting (object). He closely monitored and raised alarm bells which, in some cases, led to interventions from the CEO and Directors. The main artifacts for monitoring (object) were the Kanban board (instrument) for daily project level reporting and the reports generated by Pivotal Tracker (instrument). The major units of analysis were the user story, story points completed per week and points remaining, and unit tests completed (instruments).

> "We track everything as an individual story. So we break huge business requirements down into smaller pieces and we measure the rate at which we are doing those. We assign responsibility to those things so we see who is responsible for what" (ath_int_dev3_pg23:1).

This rate is the user stories completed per week. This is further gauged according to individual developers. However, working software components were not always considered a measurement of progress:

> "The end of the sprint does not necessarily result in a release. Sometimes there is not enough or no functionality to warrant a release i.e. there is nothing working that can be demonstrated" (ath_obs_dev2_pg2:6).

For this purpose, user stories are created for non-development tasks like architecture design and database design. Completing these stories signified progress. Testing reports (instruments) were also used as a means to track progress (object) within a project. Unit testing was automated and formed part of Test-Driven Development (TDD). Total unit tests passed and code coverage reports, served as verification that a particular user story had been tested and was ready for deployment. The Scrum Master generated these reports from Pivotal Tracker. Ideally a project team strived to maintain a consistent and healthy code coverage and unit testing rate from one sprint to the next.

The Scrum Master highlighted some aspects of communication interfaces within the boundaries of the SMME:

> "It is informal for the rest of the company. One of the nice things about having a board up like that is [CEO] can walk past and straight away say oh, there's a problem with this project or know that things are going smoothly. That's one of the great things about having that up there. Obviously the rest of the company [any other employees] can do that" ath_int_sm_pg11:6.

In an attempt to improve accuracy of reporting (object), developers (subject) consistently updated their task allocation and indicated task completion on Pivotal Tracker and on the Kanban board (instrument) during the daily stand-up meeting. Pivotal Tracker could generate graphs instantly to show project status. The development team was required to interpret these generated project artifacts. The Scrum Master explains:

> "Visual management, I think, is critical. We have the graphs that I share with the guys, the burndowns, the stories moving across the Kanban and graphs on code coverage and things like that. Those things are critical to communicate to the team this is where we are guys and this is something that needs to be addressed. But on top of that the team needs to know how to interpret that as well. Because just putting it up every day doesn't mean anything if you are not drawing attention to it and saying guys this is really not very good and needs to be worked on" (ath_int_sm_pg12:2).

These sentiments were added to by a developer:

> "The visual management system allowed us to pick up on when things where deviating very early before it became critical, and reflect on it" (ath_int_dev2_pg19:3).

The burndown graph (instrument) was a primary artifact for progress monitoring (object) and was used by the Scrum Master, developers, and by management (subjects).

> "This is an example of a burndown chart. It shows actual points against target points for the week. If the customer or someone like [CEO] needs to see progress then this chart shows that. A flat line represents a problem because that means no work done. So management will have to act and figure out what the problem is, like too much work, too difficult, look at programmer ability etc." (ath_obs_pg3:2).

Reports that indicated a negative status often triggered reflective interventions such as innovation workshops or brainstorming sessions. During these interventions the development team analyzed their actions and proposed ways to overcome the current adverse areas. This discussion will be expanded in the later subsection on learning.

The Scrum Master (subject) also maintained a spreadsheet (instrument) which was used to consolidate and report progress and financial health of all projects (object).

> "Upfront very early we estimate a total number of points for the project. And then every week we refine that. As to how much effort we need to put in for that following week, what we do there is I've got a spreadsheet where I track all of that; how many points we burn through every week, how many points we have remaining on the project and I graph that along with budget. So we see as we burning through money and we burning through points, is there a correlation? There certainly should be and then using that you can then predict how many points we can get through for the rest of the project… I've got a prediction per day of how much we expect to spend on each resource [developer] and how many points we expect to get through. So then what I can do is say at this point of the project we are above budget or below budget. I have a graph that basically shows being on budget is zero and as we deviate from budget over each day we can see that we've overspending or underspending. So that graph I keep updated. Once a week I update it with the budget figures from the previous week and report to the project manager on the client side. So financially that's how we know where we are. In terms of how we know if we going to get finished, we use the same points and show how much we got remaining and that's how much time we got left" (ath_int_sm_pg11:2).

This manual process and reporting was necessary as the client was not fully accustomed to user story points concepts. Accurate reporting often hinged on the Scrum Master's ability to reconcile points remaining against the remaining budget for a particular project. Recall that points per project was estimated by developers while the actual budget was set by CEO and Directors which resulted in a degree of mismatch.

Sub—rule—com: Developers (subject) reported on their individual progression (rule) within each project (community) during the daily stand-up meeting and by capturing details on Pivotal Tracker (rule). From a program management perspective (community), the Scrum Master (subject) had to consolidate (rule) these individual reports before reporting to stakeholders with a vested interest in that project, particularly Directors and Product Owners (subjects).

Negative reports triggered action or intervention (rule) from management (subjects) in an attempt to alleviate pressure within that project (community).

Communication of progress reports within the boundaries of Athena was mainly informal. The Scrum Master spoke of the program reporting interface between the client and the development team for one of the projects:

> "We do a project status meeting which is kind of [reporting] where we are, how much we spent, and how much we got left. This happens every Monday. So it's a telecon with the project sponsor on the client side, the manager and [Operations Director] and myself. And then every Wednesday they have a teleconference with a user representative. That goes through what we have completed, this is what we can deliver, this is what we've done, this is what we plan on doing next, do you still agree?, and is it the most important thing for you to do" (ath_int_sm_pg11:6).

The CEO highlighted additional aspects in terms of monitoring project progress at the program management level:

> "There's plenty of project metrics in terms of how the number of points is achieved that measures performance. But it's also in terms of the number of bugs picked up and it's also feedback in terms of stress levels on how the developers feel. It's also feedback from the customer on how happy or satisfied they are with what's being delivered, the interaction and then the overall performance of the team. It's a mixed bag of quantitative and qualitative measures" (ath_int_ceo_pg3:2).

For the strategic management level the aspects of customer satisfaction needed to be monitored and could not be accurately gauged from economic lenses such as user story points and budget remaining. Instead, unique mechanisms were in place for this, such as feedback reports from developers, monthly manager meetings, and sprint retrospectives.

Com—div of labour—obj: This relationship was not published due to similarities to the narrative for subject-rule-community above.

Test-driven development (TDD) was practiced at Athena and consequently progress was a measure of the amount of code that had passed testing against a percentage benchmark of code that needed to be tested. At the time of fieldwork, the benchmark for code coverage at Athena was in the region of 90%. Project progress tracking was done in real-time and done at least once a day, which confirmed the quick pace of deviation identification and correction. Reporting progress reports usually occurred on a daily basis and in some cases twice daily. With the plethora of visual and software aids in place, interpretation and dissemination of project status appeared seamless. Informal communication was frequently observed. Overall the fieldwork revealed that metrics for progress monitoring at Athena are still in a state of discovery. Team performance measurement and progress prediction in the context of changing objectives presented a challenge worthy of future research.

6.3.7 Planning

Planning practices mainly involved practices for setting of milestones for project deliverables. Planning for individual projects was done by the developers who often sought counsel from senior personnel and the Product Owner. These plans fed back to the Scrum Master who used project data to maintain a consolidated plan for all projects. In line with this the narratives that follow show planning at the project level, which then interfaced with and influenced grander plans at the program management level. This subsection will show the different personnel, project artifacts, and electronic aids that were implicated in these activities. The primary unit of measurement in planning was the user story point and user story priority. Recall that user story point represents the perceived complexity of individual constituent functional aspects of the software product, while priority was the relative importance of that user story in the holistic software system.

Sub—inst—obj: Planning typically began with the Product Owner (subject) having a vision (instrument) of the software system and its intended functionality. This was then translated into a prioritized list of user stories in the form of a product backlog (object). The development team (subjects) prioritized and assigned points (object) to each user story in the product backlog (instrument), a process they called backlog *grooming*. Every user story was then captured in Pivotal Tracker (instrument) forming an electronic version of the product backlog. The product backlog (instrument) was used by the development team (subjects) to draft a release plan (object) which set the number of sprints, and the functionality that would be created and delivered for each sprint. Although these were not set in stone at the onset they were concrete enough to warrant the start of a project.

The Scrum Master (subject) was able to broadcast plans (object) through Pivotal Tracker, visual aids and via informal communication with the developers (instruments). The product backlog and release plan (instruments) provided information pertaining to descriptions, functionality release dates and sprint review milestones (objects) for the project team, or other personnel (subjects). At Athena these existed in the dual format of physical and electronic versions which were easily visible.

Sub—rule—com: Product backlog planning for the project (community) was done by the Product Owner (rule) with help from the project team and/or senior personnel (subject). Release planning for a project (community) was done by the developers (subject) using expert judgement (rule). Estimations varied from project to project, and from one developer

to the next; for example, a senior developer may estimate a user story lower than a novice developer would. For this reason planning for individual projects was done by one or two (rule) developers assigned to the project (subject) to allow an aggregate. When needed, seniors helped estimate more intricate or highly technical functional requirements. Planning practices at the time of the research were admittedly some distance from pin-point accuracy. The following extract from the field notes highlights this:

> [Scrum Master] states that team needs to achieve 8 points today.
>
> "Are you confident with the work/tasks for today?" (Scrum Master)
>
> "That's tough" (developer3)
>
> "We need more accuracy in terms of the sprint planning tasks for the week and the prioritization of the stories" (developer2)
>
> "That's fine, we can look at prioritization but we must also ensure that stories come through" (Scrum Master)
>
> "We should concentrate on delivering value adding components to improve our poor flow rate and increase points" (Scrum Master).
>
> "We should develop performance measurement mechanisms which should include standard deviation" (Operations Director)
>
> (ath_obs_pg6:2).

The impact of ambiguity in user stories on planning was noted in some instances. This further alluded to planning not being an exact science:

> The difficulty seems to stem from the unclear/ambiguous requirements. [Operations Director] and the developer has opposing views on a particular story. They take a while to revisit the actual software components before reaching an agreement. Some of the dialogue goes:
>
> "That was not my understanding and there is a massive difference. I recall from the meeting…." (Operations Director)
>
> "But that's how we developed it in the first release" (developer1)
>
> [Operations Director] had to deliberate over this issue. This event questions one of the principles of agile namely less documentation. By just listing stories, most of the fine grained detail of the user story is in tacit form within the [Operations Director]. Traditional development would have very detailed reports/documents on requirements.
>
> (ath_obs_pg10:6).

The Scrum Master (subject) reported any deviations from the plans (rule) with the aim of trying to ensure that the different teams met their planned targets (rule) for their projects (community). Part of his program level planning involved staff allocation (described earlier).

For example, one of the projects had a tight deadline and the client was unwilling to renegotiate. Hence, part of the Scrum Master's (subject) planning was to take initiative and have some developers dedicated to that project and not removing them or burdening them with work from other projects. This sometimes presented a challenge given the small workforce of the SMME.

Com—div of labour—obj: The Product Owner for the project (community) performed the product backlog planning (division of labour). Some of the team acted as coaches (division of labour) and guided the Product Owner in drawing up the product backlog and embodied user stories (object). Initial creation of the product backlog was done by the Product Owner with assistance from the developers and Directors from Athena. The premise being that the Product Owner had intimate knowledge of his business and the required software functionality. Hence, he was better positioned to create the product backlog. This is highlighted in:

> "But most critical, is an understanding of what the customer understands to be important. Because it's all well and good delivering a project like this with all the secondary functionality and never delivering the primary functionality. As far as anyone is concerned, the project will fail regardless of it being bug free and on time or anything else. If it's not actually solving the customer's problem what's the point?" (ath_int_sm_pg12:3).

The approach described above represented a breakaway from the more traditional software engineering approach where the development team itself extracts and documents functional requirements from the client business. However, this approach was not bulletproof as indicated in the following conversation extract from observation notes:

> "[Operations Director] offers his insights into features that will improve the software even though the Product Owner has not asked or requested it yet. This is a frequent problem when the Product Owner (who is a client representative or project manager from the client business) does not fully understand their own business processes but is simply doing what they are told to do by their bosses.
>
> And the subsequent effect on the plan?
>
> "There are added specifications. What impact is there on the budget? Do we drop some functionality or do we ask for more money?" (Scrum Master)
>
> "I have shown them PivotalTracker, with the list of prioritized backlog. We then argued for functionality." (Operations Director)
>
> "Do we keep the original release plan schedule?" (Scrum Master)
>
> "This is agile. It has to be redone." (Operations Director)
>
> (ath_obs_pg35:2).

And further in the following extract soon after a product backlog grooming session that was held overseas with the client:

> "There are ridiculous things in Pivotal due to non-IT people being present at meeting in UK" (Operations Director) (ath_obs_pg37:1).

The developers assigned to a project (community) were the main architects (division of labour) of the release plan (object). The applied notion was that the developer allocated to the task was in the best position to estimate how long it will take *him or her* to complete said task. The Scrum Master had the responsibility of consolidating the different release plans (division of labour) from the concurrent projects (community) in an attempt to create a holistic, program level plan. This higher level plan was reported to senior personnel during Athena's management meetings.

> "Planning is important. You got to know only really up to a week ahead where you going, what's the vision, what is it that we need to deliver" (ath_int_sm_pg12:3).

Even at program management level, planning did not seem to stretch too far into the future. It's noteworthy that estimation of and subsequent planning for the sprints in individual projects was done by the developers allocated to that project and not by higher order personnel. This was aligned with the ethos of self-organizing or self-managing teams. But more than that, it also seemed to afford a mechanism whereby milestones were set by the actors responsible for the work; hence the potential to establish more accurate and achievable targets. The Scrum Master consolidated plans for individual projects into a larger, holistic view of projects. This viewpoint allowed for planning at the program management level. Plans were not too far reaching which created pressure on developer allocations. Although dedicated developers were in place for projects, client requests triggered change in plans. Constant planning and re-planning did distract the developers from their primary goal of trying to meet sprint deadlines. Clients were mostly well aware of the Scrum methodology and understood the ramifications of additional requests or rework. Hence, they understood the need to renegotiate target dates or permit an increase in cost so that more manpower could be added to meet targets. Overall, estimations were largely derived from intuition and expert judgement, which shaped the nature of resultant plans. This was far from the typical traditional software engineering trait of detailed upfront planning. Client involvement in planning seemed not always flawless.

6.3.8 Client liaison

Unresolved misunderstandings and missing information pertaining to software requirements can be detrimental to agile project progress due to its distinctive short delivery cycle. This is exaggerated in a SMME environment where there is a thin margin of tolerance for long wait states. To alleviate this tension, client liaison practices sought to both engage and maintain easy flowing communication between the clients and project teams. The primary conduit for communication between the project and Product Owner was the Scrum Master. In addition to this, client liaison represented the key set of practices by which clients were educated on the Scrum approach to software development, and this manifested as a key factor in establishing trust.

Sub—inst—obj: Project clients actively engaged with project teams via a variety of mechanisms. Client-developer (subject) workshops (instrument) were initial meetings where business requirements were discussed and the product backlog was created (object). Sprint reviews (instrument) demonstrated the software (object) by the development team (subject). Other practices included a mixture of formal scheduled and informal meetings, and communications (instrument) to gain clarity on functional aspects (object) and to resolve technical queries via telephone or e-mail (instrument) with client liaisons or IT staff (subjects).

The Development team and senior personnel employed a multitude of ICT devices for client liaison activities. These were contemporary technologies, such as multimedia and voice conference calls and sometimes social media based communication. The idea was to permit any form of communication that the client felt comfortable with. Face-to-face meetings were done for product backlog development and sprint reviews. However, these were limited for international clients and those project teams reverted to ICTs.

Sub—rule—com: Sometimes project teams (subject) had access to an onsite client representative (subject) stationed in the open plan office (rule) for the duration of the projects (community). This practice proved beneficial when dealing with technical queries (rule) which were otherwise resolved via Skype and telephone conversations between the onsite liaison and the client IT staff (subjects). The Scrum Master (subject) often contacted a Product Owner on behalf of the development team (rule). Not every developer assigned to a project contacted the Product Owner as they pleased; instead, this line of communication appeared restricted to the lead developer.

In a fast-paced project environment such as Scrum, frequent communication and feedback appeared to be an important commodity, as explained by this developer:

> "One of the things agile struggles to balance is operational efficiency and client experience. Athena tries to get feedback from the client as early as possible so the process can adapt to give that client the best experience possible" (ath_int_dev2_pg18:2).

Leading from the above, a strategy at Athena (community) was to ensure that the client (subject) enjoyed the experience of interacting (rule) with the project development team. This notion was referred to as *client enlightenment,* a term used by Denning (2010b). The Scrum Master highlights the importance:

> "It is important for a number of reasons. If the client sees how you work and understands and gets that good comfortable feeling there's a lot more trust. And the more a client trusts you the more you move from a client-customer relationship to more of a trusted advisor. We don't want to be a PG-Glass who you phone to fit a new glass. We want to be the guys you phone to ask what kind of glass should we fit. So then we offering a bigger, grander type of consulting role where you actually go to the client's place to help them solve their problems…Very often, especially in software development, clients don't know what they want" (ath_int_sm_pg13:5).

The Technical Director also provided insights into the nature of client engagement:

> "It's very important. When a client comes to us they don't really know what to expect. Generally they have either done software projects before and they think that it is done in this way and then we do things a different way. Or have never done software projects before and it is all brand new to them. So a lot of the client engagement is coaching them, teaching them how we do things and how they can be involved. Especially for people who have done waterfall before, it's showing them that they can be involved in a much bigger way and over time getting them more involved. That's really important for us because what we found is if the customer has more involvement they buy in more to the process and they buy in more to the software itself…The feedback we get from clients is that they enjoy the process of being involved in the development" (ath_int_tdir_pg9:1).

The CEO (subject) spoke about a different perspective of inevitable change (rule) within a typical software project (community).

> "Change always occurs. Change can occur at any stage of the project. What cannot occur is a change from trust to mistrust. So defining the relationship is very important. We need to define what the deliverables are clearly upfront and mistrust only creeps in when both parties haven't understood their responsibilities and if one party has a perception that the other party is not actually doing their part in making the project successful. That is why upfront we discuss accountability, roles, and how we will communicate" (ath_int_sm_pg2:3).

It appears that client liaison practice involved establishing a relationship of trust, clearly identified roles, and boundaries.

Com—div of labour—obj: Usually the Scrum Master, lead developer, and senior personnel were responsible (division of labour) for establishing sustainable client liaison practices (object). The Scrum Master maintained the most frequent contact (division of labour) with the Product Owner as he attempted to afford more clarity (object) for the developers working

on a project (community). The lead developer, along with one or two senior personnel (community), performed sprint reviews (object) and facilitated product backlog workshops (object). Projects (community) were not always equipped with a client representative (division of labour) for real-time query resolution (object).

Being an SMME, Athena did not have a separate marketing arm nor were there any personnel dedicated to separate marketing related activities. Potential clients usually interacted with the CEO or one of the two Directors. However, all personnel served as ambassadors for Athena.

> "People come to us so we haven't exactly been marketing. If someone should come to me and I was dealing with it then I would be accountable for ensuring that that customer is happy and gets the software that solves the right problem" (ath_int_ceo_pg1).

In some cases the Product Owner had the added responsibility of campaigning (division of labour) for the Scrum approach (object) within their own businesses (community). This involved trying to sway and encourage clients and other stakeholders to trust the agile approach.

> "We were fortunate that [Product Owner] and [client representative] very much buy into the way we work. So fortunately for us they filled almost a sort of buffer between [client's] very waterfall mind-set and our mind-set. That's helped. They planned their waterfall stuff to coincide with our agile releases so we can keep things tied up and keep things matching" (ath_int_sm_pg11:3).

Client liaison appeared to have practices that went deeper than just maintaining constant communication for the purposes of technical queries. It also encompassed the ideology of marketing and campaigning for the agile cause. This was vital in light of the strongly entrenched waterfall or classical software engineering mind-set of many software system clients. Agile teams needed to work hard to establish trust between their processes and their clients, both in their ability to deliver value-driven software systems and in their approach to developing these systems. Getting the clients more involved in the development process and, more importantly, getting them to enjoy the involvement and interaction manifested as another mainstay business strategy at Athena. Access to Product Owner was controlled but left open in the case of the client representative. Both were portrayed as active members and as change agents within a project.

6.3.9 Schedule risk identification and mitigation

There were numerous observed risks during the course of the fieldwork arising at project level, program level, and organizational level. Examples of risk included competition from other SMMEs, a general slump in the South Africa economy at the time of writing, and a

shortage of highly skilled and experienced employable candidates in the software development industry. However, the focus here was narrowed to program management risk in the context of the project schedule. The broader risks identified above, although present, were not explored further during the fieldwork.

In keeping with SMME characteristics of multi-project environment and limited resources, usually the materialization and subsequent mitigation of risk in a project had the potential for ripple effects in other projects. The most commonly observed risk was when a project fell behind schedule. Some observed reasons for this included public holidays, developers on leave, bugs, misunderstood requirements, or inadequate skill and experience levels of teams. Although risk was generally never completely avoided, practices in this knowledge area sought to minimize its effects.

Sub—inst—obj: The most frequently called upon artifact for identification of flaying schedule (object) was the project Kanban board (instrument) which was maintained and updated on a daily basis by the project team (subject). The Scrum Master (subject) generated burndown and budget deviation graphs (instruments), which served as indicators for project schedule risk and budget risk identification (object). Senior personnel, Scrum Master, and any of the developers (subject) could identify lagging schedules or slow progress (object) directly and easily from these project artifacts (instrument). Pivotal Tracker (instrument) facilitated by generating graphs and estimates on completion time (object) on-demand for the Scrum Master and project teams.

Incorrect user story point estimation was a frequent risk as these formed the basis of accurate scheduling and budget. This was the focus of discussion in the earlier section on cost estimation. In context of risk, story point estimates were usually off by a small margin at the beginning of projects. Their accuracy improved with experience as the project progressed. The following narrative during a daily stand-up meeting bears proof of this nature:

> "Why is a story in WIP for 4 days?" (Scrum Master)
>
> "Because we have changes to Siebel that are not being communicated." (Operations Director)
>
> "We need to learn lessons, why estimate indicated as 1 day but took 4 days." (Scrum Master)
>
> "Technical lessons. 1st time with new tech takes longer but then gets easier with more experience" (Developer1)
>
> This is an interesting point raised by the dev, how does this influence scheduling and estimation when new technology is being used in a project?

> "There is rework in Siebel, but we can't track that progress. We need to show rework" (Operations Director)
>
> The team agrees on the need to show why a story is on the board for so long. The suggestion is to use additional stickys [post-its] to show why.
>
> "We need to allow the customer to understand why project late, which can then allow them to make a decision. For this we need information to show why. Most customers are accommodating in changing project schedules. A few are not, but most are." (Operations Director)
>
> "We only know when we are out of control at end, but here we can catch it early." (Operations Director)
>
> (ath_obs_pg23:2).

Mitigation for inaccurate mitigation was attempted (object) via practices; for example, brainstorming sessions, innovation workshops, and sprint retrospectives (instruments) which were carried out by the project teams (subjects).

However, there were also meetings held prior to the onset of the project. The following two interview snippets showcase this:

> "At the beginning of the project I sat down with the team and we brainstormed everything that we imagined to be a risk on the project. So a bit of risk assessment. We then looked at how critical that risk would be [if] it turned out to be true and what impact it would have on the project" (ath_int_sm_pg11:3).

Post sprint review, retrospectives provide insights into experienced risks specific to sprint.

> "We have weekly retrospectives where we brainstorm ideas and try and come up with solutions to fix problems as they occur" (ath_int_dev3_pg22:5).

The following snippets from observation notes lists topics for discussion for a retrospective that was held after the second sprint for one of the projects. The personnel that raised the topic are also listed.

> - Over deadline by seven days [Scrum Master, Operations Director]
> - Over budget USD 5900.00 [Scrum Master]
> - Slow skill transfer [Developer 3]
> - [Client ERP] being offline resulted in delay [All]
> - Under estimation of points in UI [Developer 2]
> - Lack of visual cues [Developer 3]
>
> (ath_obs_pg15:4).

Client workshops (instrument) that were held between the development team, client representatives, and Product Owner (subjects) served as mechanisms for risk alleviation (object). Close and early client involvement facilitated in lessening the impact of ambiguous and misaligned expectations in the delivered software system to a certain degree, which in turn reduced the risk of falling out of schedule and budget. This notion was the subject of the previous subsection.

Sub—rule—com: Within each project team (subjects) corrective actions are taken (rule). The Technical Director explained:

> "Essentially you try to get more productivity out of the team for the same cost so that you can get back into the black. What you do there is look at what you are doing and how you can improve it. So you spend a lot of time analyzing the way we [are] working, what's not working, what's slowing us down. That's what retrospectives are partly and we had some specific workshops to figure out how we can speed up and where the bottle necks are... Like on [project4] with just me and [developer5], I'm watching the budget burndown and I'm watching the points burndown; what you want is the points to be the same as the budget burndown. If that's not happening and the budget is burning down faster than the points then I've got to go to [developer5] and ask, look what can we do? Either way we have to reduce the scope of the project or we have to figure out how to develop faster. The thing with that problem is that you can't add people because adding people will increase the budget. So adding people might be something you would do to bring it in a bit faster but it doesn't mean you will bring it in cheaper" (ath_int_tdir_pg7:4).

From the narrative above, the SMME characteristic of a limited pool of resources appeared to influence risk mitigation practices. Contrary to the above belief that adding more developers will increase project costs, in some cases the Scrum Master or senior personnel (subject) did intervene by adding more developers (rule) to a project (community) for a certain period of time. The reason for this was noted on one of the projects that required an intricate UI component:

> "The highest risk that we came up with was the user interface (UI). Largely because we were bringing in a UI team who were great at user experience (UX) but did not know the technology. So there would be a large up-skilling coming in there. So to mitigate that we brought a developer, that wasn't on the project, in for a week to pair with the UI team and teach and coach them" (ath_int_sm_pg11:3).

Risk mitigation sometimes occurred in the form of client and Scrum Master (subject) negotiation (rule) over functionality, budget, or delivery dates for a project (community). The catalyst for this type of intervention was, more often than not, misunderstood or miscommunicated functional requirements. At Athena, the prevailing ethos was one of providing value to a client business and not simply a software system. As a form of risk avoidance, attempts were made to define this value early on in each project. The CEO elaborated on these attempts:

> "We need to understand [from our clients] what business value this piece of software is going to add and the value proposition must be clearly defined. If the customer has been unable to define it or if the customer believes that the software is miraculously going to solve a whole lot of underlying issues that are systemic to his organization, then it's our duty to point this out" (ath_int_ceo_pg2:2).

In some cases the team elected to practice pair programming (rule) for more complex user stories, which usually slowed down overall project progress (community) since two developers (subject) were working on one user story. However, if the incomplete user story was causing a major disruption within a project then this expenditure was motivated.

In some cases the project team (subjects) held a workshop (rule) to identify possible causes for falling behind schedule and to formulate possible resolutions (rule). These workshops were held periodically but were sometimes called for immediately (rule) by the Scrum Master or Directors (subjects) of the project (community). This practice aimed to provide the project team with sufficient *room* for analysis and innovation towards risk mitigation. The Product Owner or client representative (if one had been assigned by client) may have also been engaged in the event of requirements uncertainty.

Risks appeared to trigger different recourses. Sometimes it resulted in the training of developers and other times it was merely a reminder from the Scrum Master "to keep this risk in the back of your mind so it doesn't occur" (ath_int_sm_pg11:3).

Athena (community) employed a handful of developers (subject) on a contract basis (rule) and these developers were not usually in office for the entire work week. This did affect schedule, especially when their timetables were not fully disclosed to the team. A conversation between a contract developer and a full time developer was noted during observation:

> "I won't be in tomorrow" (developer4)
>
> "But I counted and planned on you being here for the whole week" (developer1)
>
> The much broader issue of employees working on contract arises once again. It would appear that this is now affecting the schedule of the project
>
> (ath_obs_pg13:3).

There were also risks that had an effect on project schedule but were not easily controlled. One such risk was labelled team dynamics. The Scrum Master explained:

> "One of the very important things is team dynamics. But this is a very difficult thing to quantify. But a team that understands a common goal and works together. Through no fault of the team we found that to be a problem in early in [project1]. We weren't gaining any traction and that was

> largely due to communication, due to mind-set of you do UI we do dev. Things like that...On top of that we had some staff in [project1] on their first project, that were new to [Athena], hadn't worked together, and hadn't kind of known how we do things" (ath_int_sm_pg12:1).

A developer added:

> "A main cause of the hold up in [project1] was the lack of understanding how the team had to gel together" (ath_int_dev1_pg15:4).

Another example that surfaced during an interview, but which was not one that was witnessed during observation, was the risk of losing knowledge. The Technical Director provided insights:

> "Developers are not cogs in a machine where if one breaks you can replace it. If a developer leaves or falls sick you are left in a pretty bad situation. There's a lot of knowledge in their heads" ath_int_tdir_pg7:4.

With the typical fast pace of Scrum projects, combined with SMMEs appetite for presentation of expertise and constant improvement of process, mechanisms to support long-term organizational learning were few at Athena.

Then there were those schedule risks that Athena had no control over. An example was the heavy reliance on client employees for system interface and integration information. There was also the observed case where the overseas client on one of the projects suffered a major failure in their database centre which caused a large amount of wait state on that project. Nothing could be done in that case except wait for resolution on the client side. At one point in the fieldwork many public weekends also affected project schedules.

Com—div of labour—obj: Being an SMME (community) the entire workforce (division of labour) played an active role in risk identification and mitigation (object). Developers identified brewing disruptions in project progress during daily stand-up meetings and altered their course of operations accordingly. The Scrum Master (division of labour) worked towards having the most up-to-date holistic image of progress, schedule, and budget (object) across all projects (community). He reported on deviations and intervened when the situation escalated.

Schedule risk stemmed most frequently from inaccurate user story estimation. Athena maintained a strategy whereby the developers were encouraged to take ownership of the project, including its embroiled risk to schedule. This was part of the development or up-skilling of staff and a move towards the ideal of self-managing teams. Having developers learn to mitigate risk was a risk in itself for Athena, but it appeared a risk the senior personnel

deemed worthwhile. It was deemed a mechanism to build commitment in the developers to generate the work, because of the accountability and the control they are given. Limited resources had an important influence on the risk resolution strategy. For example, adding additional developers in the form of pair programmers often impacted on other projects. Instead, ways were sought to alleviate risk without simply relying on adding more manpower. Team dynamics and practices for establishing a *gelled* team was deemed important. SMME reliance on contract workers did interfere with schedule. There was a general lack of explicit learning practices; nonetheless, techniques such as sprint retrospectives allowed for visibility of schedule risks being experienced.

6.3.10 Aligning practices with business strategy

A deep enterprise level investigation into types of business strategies, implementation, and evolution, was beyond the scope of this thesis. Instead, the field work honed in on instances of interfaces between program management and business strategy. It was noticed that business strategies were impressed in the different projects through mentorship and coaching, in an attempt to ensure all employees practice these principles. Although there were many observed business strategies in place, this thesis focused on self-managed teams, client interaction and involvement, and teaching and coaching development teams. These were deemed key in the context of program management.

Business strategies are in effect rules or norms that influence the practice of program management. Hence, this subject—rule—community relationship is explored first.

Sub—rule—com: Self-managed or self-organizing teams (rule) was practiced throughout Athena (subjects), in all projects (community). The following narrative is a response to self-management in the context of Athena:

> "I think that's kind of the wrong terminology. I'd rather say self-organizing. To create that you still need the project leader but their job is fundamentally different to the traditional manager in that the leader is there to facilitate. It firstly assumes that people in the team are motivated to do their job. To see the project succeed and are capable to fulfil their job" (ath_int_dev1_pg17:3).

The Scrum Master's sentiments provide an interesting perspective:

> "Self-managed teams is a utopia state. It's where everyone would love to be and where every team wants to be and where every manager wants their teams to be. But it doesn't just happen. You get that only when every team member truly understands what that means. They all buy into the goals, the vision of what the team is trying to achieve. They must have a common vision. There is no real management from the outside because the teams correct themselves, they look after themselves. When I say team I mean a customer representative, so you have a Product Owner or a project sponsor who knows exactly what the customer wants and drives the requirements, you got a Scrum

> Master and you got your team members. So in order to have a properly self-managed team you need to have all the correct role players in place so you got all the inputs and all the outputs" (ath_int_sm_pg12:4).

These sentiments are mirrored by the Technical Directors:

> "We have a belief that if you have the right structures, the right processes, and the right forums in place then you don't need to manage as much. So you try and build all these things to allow people to self-manage, to self-correct. We haven't got it right. As management we are still correcting things. So we still feeding, still learning on what we can do to improve those processes" (ath_int_tdir_pg9:2).

Both responses infer a cautious attitude when describing self-organizing instead of an outright, detailed implementation response.

Client interaction and involvement (objective) encompassed practices that ensured that management (subjects) chose the right type of customer (rule). The CEO elaborated:

> "We are very selective in terms of the customers that we choose. IT is a difficult terrain in that networking gets lumped in software and people are not always aware of the complexities involved in software development. So in terms of client enlightenment, it's in terms of informing the customer of the life cycle, the methodology, the risk and the way we work and the fact that they must be willing to engage in a partnership where both parties are mutually respected. If they are unwilling or if we feel the customer is not mature enough to handle such a relationship then we would walk away from the project" (ath_int_ceo_pg1:4).

Client involvement was important for day-to-day operations:

> "We want them to be involved as much as possible. The amount of information and help you get from having the client present is incredible because of the immediate feedback or response" (ath_int_dev1_pg16:3).

> "One of the success factors was realising early and getting feedback early and trying to understand how far off the mark we were in terms of providing value. That gave us a very accurate measurement of how far we are and what we needed to do" (ath_int_dev2_pg18:4).

However, this involvement should be a rewarding experience from the client perspective:

> "Client enlightenment is of massive importance because as a company we are offering a service to a client and if our values individually can be aligned with that, that would be hugely beneficial because we don't write code for the sake of writing code. We write code for the sake of benefiting the company and seeing those people in that company really flourish…Clients need to feel part of it. They need to feel that they've been taken through this journey to make their company do better and flourish" (ath_int_dev3_pg23:3).

Duality of immediate feedback for improved project progress and a perception of a fruitful engagement in a project stemmed from the strategy of only engaging in business with a client that believed in the usefulness of the agile approach.

Coaching or mentoring (rule) was a practice that manifested in almost all areas (community) of the project level. Coaching was done by senior developers, developers, Directors, Scrum

Master, and CEO (subjects). This type of interface was possible in an SMME like Athena due to a more flat organogram with a small workforce and closer relationships. The following quotes highlight the role of coaching:

> "Coaching is important. My role is, if I find the time, coaching. The more the team understand the big picture in terms of exactly what their responsibilities are, exactly who should be doing what and why it should be done, and why it should be done in a certain way then the more you can stand back…For us [Athena] it is largely about coaching and learning" (ath_int_sm_pg12:5).

> "The first thing to do would be to understand from the team's perspective where they see themselves in terms of performance and to unpack and understand the issues and to help them problem solve or facilitate their problem solving so the team owns the issue. It doesn't help to solve the problem for the team. So a lot of time is spent on coaching the team in understanding what are the issues and what are they going to do about it" (ath_int_ceo_pg4:2).

> "We encourage people to do training and practice their craft. And to try and get everyone to pair up. That's one of the ways in which we get the seniors to teach the people that are less experienced" (ath_int_tdir_pg5:4).

> "So far we try to match skills with project and we try and push people into areas they haven't worked in before. You want to make a team with people that have worked on that similar project and those that haven't; so you always trying to skill people all the time so they keep learning" (ath_int_tdir_pg9:4).

Sub—inst—obj: The practices for promoting these values and business strategies (object) appeared to involve primarily informal and formal communication, daily stand-up meetings, workshops or sprint retrospectives, and brainstorming sessions (instrument). Senior personnel and in some instances the CEO (subject), motivated the choice of a certain recourse within a project by referring to a business strategy; for example, client enlightenment or forcing upskilling of an employee.

Com—div of labour—obj: Scrum Master, senior developers, and Directors assumed the role of coach and mentor (division of labour) preaching business strategies and company values (objective). The CEO is primarily responsible for envisioning and setting up these strategies at enterprise level (community). Developers (subjects) working on the different projects (communities) tried to implement business strategies during daily operations and were responsible for the pragmatic implementation of the practices summarized in this subsection.

Self-organizing teams seemed an ideal. Practices were in place to allow for more complete client engagement. An important strategy was to choose a client only once they exhibit a willingness to learn and provide the level of interaction required by agile. Client engagement on a day-to-day basis was important to project success. Hence, clients need to appreciate the process and freely participate in and *enjoy* the experience. Knowledge transference within the SMME was a frequent observation at all levels. This was facilitated via Scrum practices and

by coaching and mentoring. Tacit knowledge transfer was frequently observed with little documentation. However, the small collocated workforce of SMME alleviated pressures on knowledge transfer. Employees were frequently forced to solve project problems on their own and frequently shepherded outside their comfort zones.

6.3.11 Learning practices

Learning is by and large an integral part of most business' operations. Although this fieldwork did not set out to ascertain the amount of learning taking place, it did identify practices that both fostered and promoted learning. Project level practices that aimed to promote learning included sprint reviews, sprint retrospectives, and daily stand-up meetings. There were numerous artifacts that promoted learning in the sense of project schedule awareness, including visual aids like the Kanban board, progress and budget graphs and charts, and Pivotal Tracker. Learning at the program management level typically involved the transfer of lessons learned and experience between different projects over the course of time. What were the major practices that fostered this form of organizational learning?

Sub—inst—obj: The Scrum practices in place in support of this organizational learning (object) were sprint retrospectives, sprint reviews, daily stand-up meetings, and brainstorming sessions and workshops (instrument). These practices were in place for the team to critique and improve on their operations within their projects. The perceived importance of these practices is highlighted below:

> "It's better to take 2 hours and understand what the problem is instead of basically producing 40 hours of useless work. That's our philosophy. We actually try to encourage people to think before they do" (ath_int_ceo_pg4:1).

> "So you look at what you doing and how you can improve it so you spend a lot of time analyzing the way we working in terms of what's working and what's not. That's what those retrospectives are about. Also have specific workshops to figure out how we can speed up, and try to find out where the bottle necks are" (ath_int_tdir_pg8:1).

Although conducted mostly by project teams, the Operations Director, Scrum Master and in some instances, the Technical Director, tried to attend these knowledge transfer sessions for two primary reasons: First, to try to impart their knowledge and experience to the team; and second, to educate themselves on challenges being experienced in the *front lines*. Their opinions and report back was expressed (object) in the weekly manager's meeting (instrument) which was attended by the CEO, Scrum Master, and Directors (subjects), as well as in a monthly Directors feedback session attended by all staff. During the manager's meetings (instrument), Athena's operational processes and business strategies were critiqued

(object) in light of the current situation, while the feedback session served as a feedback loop to the developers. The SMME nature catered for these lightweight approaches.

An internal social network served as a supporting channel for knowledge transfer. However, this appeared more geared towards technical project level work and sharing of expertise and lessons in that context.

> "We also have an internal social network caller Yammer. It's purely for [Athena] and on there people share ideas. They might have got stuck on something and they share what they did to solve it. Or if a project went really well, then the team can post what happened. For example, really good feedback from the customer" (ath_int_tdir_pg8:4).

Com—div of labour—obj: The more experienced personnel tended to coach others on operational issues, business strategy and in some instances on business contexts of a particular client. All personnel (community) were encouraged to e-mail or speak directly (division of labour) with the CEO regarding their ideas for improvement in operations (object). But a narrative from the CEO revealed an undercurrent:

> "Basically it's the staff telling me in terms of what they view as important…Value systems, their lives, where do they see themselves, what is it that they want out of a career…You looking at the organization through many lenses" (ath_int_ceo_pg3:1).

Given the small workforce (community) of the SMME, knowledge was usually transferred (object) through formal practices and informally via coaching and mentoring. All developers (division of labour) actively participated. The effectiveness of this feedback loop between manager and developers was questioned in the interviews:

> "That's a very difficult question. I would probably say by looking for results…For example [Operations Director] jumped into [project1] because he could see things going wrong through his experience. So he said ok, cool, lets jump in and see if we can learn from it" (ath_int_sm_pg13:3).

> "We do informal project reports where we meet once a week to discuss how the projects are going. There's a lot of informal discussions that take place at lunch, coffee breaks etc. That's why we have that separate area. It's a place for people to chat about what they are doing" (ath_int_tdir_pg8:3).

The CEO was asked about her role in supporting organizational learning:

> "That entails basically the reviewing and identifying lessons learned throughout the organization. So that also occurs at a management level. We have a management meeting every Tuesday and we have a Directors feedback session once a month to all staff. So it's an opportunity for everyone to reflect on growth and movement of the organization. The things that we have done well and the things we could do to improve and actually become better... It's reflective and includes many forms of feedback. It could be client feedback, employee feedback, managers reflecting and saying ok, we could have done this better if we handled the situation differently" (ath_int_ceo_pg3:1).

Management provided support (division of labour) such as budget and time for training (object) for developers or managers (community), or granted time for trouble shooting workshops (object) within individual projects.

> "We've always dedicated 20% of our production time towards innovation. So a lot of learning happens during that time" (ath_int_ceo_pg4:2).

> "At the beginning of the year we decide on what training is needed. These are soft skills as well as technical skills. It is my job to lobby for and find funding and help select the courses that would be helpful" (ath_int_ceo_pg4:3).

Sub—rule—com: Rules in effect at Athena (community) aimed to foster learning and knowledge transfer and included mandatory Scrum practices (rule), compulsory attendance of the development team (subjects) at these meetings, and an observed norm of attendance by Directors (subjects). There was a culture of balancing learning and productivity (rule). For example, developers (subjects) were given tasks outside their established areas of expertise, or developers experimented with new technologies on their own free-will. Although not a rule, it can be considered a norm and personnel continuously pushed themselves out of their *comfort zone*.

A large amount of experience and lessons learned appeared to be tacitly held. Being a SMME, this knowledge was transferred between personnel and the CEO through both formal and informal mediums. There were limited instances of knowledge being stored explicitly for future reference:

> "We don't really do that. There is no documentation of lessons learnt. Retrospectives are done but we don't document the retrospectives. Retrospectives action and bring up the learning, to action some learning but there's no documentation" (ath_int_dev1:pg17:2).

The following narrative signifies a potential negative effect of this course of action:

> "If a developer leaves, you left in a pretty bad situation then. Because a lot of knowledge is in their heads. Especially on smaller teams" (ath_int_tdir_pg8:2).

It was evident that the small workforce and flat hierarchical structure facilitated in the transfer of knowledge between personnel involved in past projects. It was noticed that documentation was not done mainly due to strained resources. The effects of this over the long-term remain unknown. On numerous occasions it was clear that Athena was in a constant state of evolution. Trial, tribulations and learning have blended into everyday routine and manifest in almost all practices. Senior personnel instilled in all development teams that this process will be a never ending cycle. The CEO provided the means for training. The

commitment to this guiding philosophy resonates in the following remark made during a brainstorming meeting by the Operations Director:

> "[Athena] employs people that are intrinsically motivated. If 95% are intrinsically motivated then fire the 5% that are not. Its management's job not to put structures in place for the 5%" (ath_obs_pg21:6).

Having now completed an examination of the Athena site, the next section focuses on the second case study site, Minerva.

6.4 Analysis of Minerva case study data

The subsections that follow present the program management knowledge areas and practices that were identified at Minerva. As in the case of Athena, observed knowledge areas were restricted to staff allocation, coordinating activities, cost estimation, human resource management, infrastructure design, progress monitoring and reporting, planning, client liaison, schedule risk identification and mitigation, business strategy alignment, and learning.

The practices associated with each of the identified areas are now described in more detail. AT was employed as the data analysis technique.

6.4.1 Staff allocation

The practices for staff allocation at Minerva included mostly assignment of personnel to new and current projects. These assignments also established roles and responsibilities for developers for a given software project. Although automated project management support tools, in the form of FogBugz, aided in this activity, the small workforce also helped to maintain acute awareness of current assignments.

Sub—inst—obj: At Minerva staff allocation (object) was a collaborative effort between the Commercial Director, Technical Lead, and Scrum Master (subject). Developers assigned to a project were flagged on FogBugz (instrument). FogBugz facilitated the staff allocation activity (object) by providing a holistic view of developer-project assignments. A less technical instrument was short ad-hoc meetings (instrument) held by senior staff (subjects) where allocation of staff was deliberated (object). This was possible due to the small workforce and easier awareness of current allocations.

Sub—rule—com: Roles and responsibilities for each project were delegated to developers (subject) at the onset of a project (rule). Two developers (subject) were initially assigned to a project (rule). Developers (community) were then informed about the assignment (rule) by

the Scrum Master (subject). Soon to be available, or least busy developers were picked for a project. Available developers or idle developers were rarely seen during the fieldwork. A developer's (subject) experience with technology (rule) and with a certain client (rule) also had an influence on the assignment (community). Developers did not have absolute choice in their assignment. A recently employed developer provided her view on how developers are selected for a project:

> "I'm not too sure how they selected. I think so on the size of the project…and the complexity of the project" (min_int_dev4_pg16:4).

There was one reported instance of a senior developer who requested to be moved to another project:

> "Guys have been moved around depending on whether they've been enjoying the work or not. As an example, I was on a project that I wasn't enjoying and I made [Scrum Master] aware of the fact and my focus was moved. So it's not an active process, it's more an availability issue" (min_int_dev1_pg11:2).

Staff allocation at Minerva was quite different from that at Athena. Here a two person project team approach was implemented. The Scrum Master and a developer explained:

> "The strategy is to have a technical lead and to have a pool of resources who are available to assist on any given project that's under pressure at the time. We had the structure before where we had 2 to 3 guys ring-fenced to work on a particular project but it's difficult to employ that type of strategy on projects that have got a very quick velocity. Because you end up finding that you have to pipe enough work for three people, you know, and that may not always be possible because you are awaiting feedback from the customer or the clients allocated to that team don't have any worked piped for the next month or so and you sit with a big problem" (min_int_sm_pg6:4).

> "It's been very haphazard. I've been here for three years now and we've kind of jumped around different ways of doing things. We went from anybody being picked up kind of like a black-box working on random work. We attempted larger teams which, in my opinion, was a complete failure, it just became too fragmented. And then we reverted to smaller teams, pretty much teams of two working in isolation on sprints. Generally we try to keep to the same product because the transfer of knowledge is expensive" (min_int_dev3_pg13:2).

Alleviating waiting states and knowledge transfer were the main motivators for this two man project team approach.

Com—div of labour—obj: From the available (or least busy) pool of developers in the SMME (community) the assigned lead developer and the second developer (division of labour) conduct the development work (object). One of them was assigned the role of lead developer (rule) while the second was designated the "coding buddy".

> "That is why we adopted the strategy of single tech lead with a coding buddy who looks at the code but the coding buddy does not have to be executing on the sprint and can be working on another sprint as tech lead on another project" (min_int_sm_pg6:4).

The coding buddy system essentially resembles the pair programming technique. However, the team could grow in size as the project need changed. Also, for larger projects additional developers could have been deployed to assist with maintaining velocity.

> "For a small project, there could be just a lead developer and a buddy. For a larger project, there will be a lead developer, buddy, and x number of contractors" (min_obs_tl1_pg6:5).

From a more risk-orientated perspective, two developers served as mitigation:

> "We have attempted to stop one person working on anything purely because if that person disappears it becomes so difficult to maintain. So we try to have at least two people on every sprint and there'll be a code review which involves somebody else at least so there's exposure to every bit of code that's written. But, from an efficiency point of view we try to do everything in pairs but due to time constraints we may have to ramp up but we found two people to be quite an efficient way to do things" (min_int_dev3_pg13:2).

Developers did not appear to have absolute choice in their project assignments. They were mostly chosen based on their skillset, prior experience with similar software or having worked with the same client previously. However, this was balanced with trying to provide all developers with exposure to different clients and technologies and avoiding developers working in black boxes. Being a SMME, management at Minerva had an acute awareness of the developer's skills and expertise. Minerva also demonstrated an interesting approach to team allocation, namely one of two person project teams that grow on-demand. The consensus was that this approach diminished long wait states and facilitated knowledge transfer from experienced to less experienced developers. Although not stated in conversation during the fieldwork, assigning developers to the same client also facilitated with client relations.

6.4.2 Coordinating activities

Practices in this knowledge area helped to coordinate software development activities in ongoing projects. These practices mainly oversaw planning of tasks, allocation of tasks, and feedback on task progression. The user story was a key metric for these practices.

Sub—inst—obj: Every user story was stored on FogBugz (instrument) which helped developers (subject) coordinate their activities (object) by providing indicators of outstanding work and deadlines. Daily stand-up meetings (instrument) performed by the developers (subjects) served as the vehicle for coordinating activities (object). FogBugz and daily stand-up meetings (instruments) helped the Scrum Master (subject) to track progress of the different projects (object). There was no Kanban board in use. The user story status was a key

indicator for work to be done, work completed, work awaiting testing, and work in progress for each project.

The two developers (subject) assigned to a project coordinated their project activities (object) through both formal and informal communication (instrument), a process that was unsupervised. A developer enthused:

> "[…] we get given a sprint with autonomy to do what we like, as long as we get it done in time" (min_int_dev3_pg15:4).

Sub—rule—com: Being a SMME (community) with limited human resources, Minerva's policy was two developers per project (rule). Coordination of project activities (community) was done entirely by collaboration (rule) among the assigned developers (subject).

> "I think I mentioned in one of our earlier meetings, case allocation is not mandated by myself or the Product Owner. It's actually a self-managed thing by the team of developers working on the milestone. Typically the guys would do it themselves based on areas that they are comfortable with, they believe they can execute most efficiently. And again I think the driving philosophy in that is through the stand-ups we have on a daily basis tracking budget. So they are always aware of the time they got available to them and I think that in itself allows them to make the decision as to which person will pick up which cases" (min_int_sm_pg6:1).

Developers (subject) chose their own activities based on their expertise (rule) and their willingness to venture into new areas of development or gain exposure to new development technologies (rule) within the assigned project (community). The generalist nature of the workforce within a SMME was demonstrated from time to time when developers (subject) elected varying development tasks (rule) within the project (community).

The Scrum Master, Technical Leads, or Commercial Director (subject) could force certain coordination decisions (rule) within a project (community). For example there was one observed instance where the entire work force was asked to do work on just one project for one day. Although this action did seem to interfere with plans for the other projects, it was necessary to improve on the lagging schedule of that one project. The Scrum Master defended this course of action:

> "There are instances where there is some intervention from the management team. But, that's typically only in an instance where an unplanned event has occurred. For example, a resource being ill for a few days and there's a fixed deadline, then we would do a manual reassignment and possibly pull in another resource to assist" (min_int_sm_pg6:1).

Com—div of labour—object: Within Minerva (community) the Commercial Director, Scrum Master, Technical Leads, and developers (division of labour) influenced coordination activities (object) for the greater good of program management. The developer pair in the

form of a lead developer and coding buddy was responsible for coordinating inter-project activities and reporting progress during the daily stand-up meetings. The Scrum Master monitored and reported progress (division of labour) across projects (object).

Interference within planned project activities occurred when management sighted a project that was dropping behind schedule and they then made an executive decision to add more contractors. Besides this, there is no other observed influence from management on the planned project activities. The developers assigned to a project planned and distributed work unsupervised. FogBugz and the daily stand-up meetings played a key role in creating awareness of current coordinated activities and related progress. Developers assigned to projects selected activities that mostly existed within their realm of expertise or sometimes ventured into uncharted waters to learn new technologies and improve their craft. Ultimately, developers coordinated their own tasks which fostered a sense of ownership and responsibility for the project.

6.4.3 Cost estimation

Cost estimation practices primarily involved monitoring and reporting of expenditure across projects. These practices focused on providing a measure of budget remaining for a project and for constant reporting to create sufficient awareness so as not to exceed the allocated budget. The user story was the primary unit of measurement.

Sub—inst—obj: This activity depended extensively on project management tools. At Minerva, FogBugz and Timetracker (instrument) were used by the developers and the Scrum Master (subjects). This snippet from the observation notes explains further:

> FogBugz project support software has a built in heuristic algorithm that calculates the projected time for completing a case per developer. The system *learns* the developer's ability by tracking their time of completion against effort required for each story. It must be noted that a developer's ability to estimate time for completing a user story improved with experience. Also, the developer's skill improved over time which means he/she will need less time to complete a user story. By using sophisticated algorithms (such as, probability settings and Monte Carlo simulation) FogBugz used past performance data for each developer for evidence based scheduling (ebs for short) and generated reports for predicted completion dates. Developers logged active and completed work on FogBugz as they go about their daily activities. Timetracker (instrument) was in-house developed software that interfaced with FogBugz. Timetracker had a database and algorithm to determine time for case completion (min_obs_pg5:1).

Minerva also sold Timetracker as an off-the-shelf project management software; hence the proprietary nature of the software prevents deep analysis of its inner workings. The Scrum Master relied on the reports from FogBugz and Timetracker to measure progression of different projects and to calculate the remaining budget for projects. The Scrum Master

(subject) maintained a spreadsheet (instrument) of project data of concurrent projects that was communicated via daily stand-up meetings or when requested by management (object).

Sub—rule—com: Within the context of Minerva (community), the developers (subject) were responsible for assigning effort to a user story in hours (rule) and for logging in the start time for work on a user story in FogBugz (rule). The Scrum Master (subject) monitored progression of the various projects and took careful note of the projected time remaining for each project. Similar to the relationship above, this projected time was generated by FogBugz.

There were two models (rule) of estimation practices at Minerva namely:

> "Fixed scope and fixed cost where it is relatively easy to allocate cases and predict time. And, time and material where it is slightly more difficult to predict time, future work, and tracking is difficult. Here we normally use billing per hour" (min_obs_sm_pg5:4).

In the instance of fixed scope and cost projects (rule), a project delivery schedule was drawn up and a fixed price derived for the complete software. Here the time frame was specified and the cost agreed upon before project work began. This seemed suitable for projects where the requirements are clearly agreed upon. Developers (subjects) coded the required user stories within the sprint (rule). In the event of the software being delivered after the schedule date, profit was lost. In an attempt to alleviate or prevent this occurrence, more developers (or contractors) were sometimes added to the project (rule) from the pool of developers at Minerva (community). At other times a later delivery date was lobbied for with the client.

When client requirements were dynamic and not well understood at the onset of a project, the time and material cost model (rule) was used within the project (community). Developers were the proverbial material (subjects) and an hourly rate (rule) was agreed upon with the client. The client was then billed at the end of a period of time (for example at the end of the month) for any work done on the project. Time and material projects were usually long-term and lasted for an average of 24 months. No long-term projects reached conclusion during the period of the fieldwork. Project cost was determined by the amount of hours spent and the number of developers used.

Clients were billed for work done on the software product only.

> "Here we look at the actual time that was spent […] there is no interruption ratio that is applied" (min_int_sm_pg8:5).

Interruption ratio refers to the practices that did not directly involve development work; for example, daily stand-up meetings and sprint planning sessions. Although the Product Owner

and developers (subjects) may have been carrying out these activities during a sprint, the client was not charged for this (rule). That said,

> "The client essentially gets billed for everything, it's just a question of whether it visible to them or a hidden cost. So, essentially we, in any business, have this concept of interruption or non-productive time. You would have heard us talk about this interruption ratio that we utilize. That is derived based on metrics of where time is logged to specific cases in FogBugz or whether it's logged to planning meetings, stand-ups or whatever the case may be. […] For example, if we working on a sprint for customer A and that sprint lasts two weeks and during the course of those two weeks there's stand-ups that happen every day, there's dev meetings that happen twice, there isn't a case that exists in customer A's sprint that the dev will log their time to. […] Instead what happens is, when we get to the end of the sprint and it is a time and materials sprint, we will look at the actual time that was spent and what the interruption ratio was for that two week period for example, and we'll divide that actual time by the interruption ratio. The figure that we get out is the adjusted and that adjusted time is what we bill the client for" (min_int_sm_pg8:4).

However, the client was billed for design or business analysis meetings. This bears reference to the hidden cost early in the above excerpt.

Com—div of labour—object: The person primarily responsible for cost estimation (object) was the Scrum Master. Developers set estimates for stories at individual project level and logged in completed stories while the Scrum Master was responsible for monitoring and feedback at the program management level (community). A senior developer described this:

> "When we get a set of cases together for a sprint we'll have two or three developers sit down together and we try to use a card system with t-shirt sizes. We'll have a look at a case, discuss the requirements and each developer will hide their estimates and then show it at the same time. If there are differences then the developers will discuss among themselves if this is a harder or easier task until there's some kind of consensus reached. So that will be a broad level t-shirt size for the story. When the developer starts on the task, he will need to assign a specific hour estimate to the task and that will be used to track their estimation history over time. Once an hour has been assigned to a task and time has been logged against that task, the task can't just be re-assigned to another developer because now there's an estimate history on that. So in that sense there is little bit of inflexibility but we work around that by creating a new case" (min_int_dev1_pg11:4).

This narrative pointed out the dependency upon FogBugz for estimation and, in turn, budget usage reports and forecasts. Before creating a new user story for reassigning the task, techniques such as pair programming were usually commissioned.

Accuracy in user story estimation improved with the passing of time on a project:

> "There is always going to be a degree of learning curve and still quite a risk in terms of having to base an entire product on delivery of a system we haven't necessarily had experience with. So again your estimates and everything that goes down is very subject to how quickly and how accurately you can learn the system. What's quite interesting is in the early stages the risk is proving quite high because there's a lot of learning so our estimates have been off but the more we learn we're realising that the system actually caters for this extremely well. So further down the line, things that we over estimated are gonna be done a lot quicker. So it will all balance out" (min_int_dev6_pg25:2).

Project teams were usually not able to one hundred percent accurately predict user story estimates. However, Scrum allows for real-time adjustment and reporting which absorbed these imbalances. This is an important characteristic for SMMEs due to their low tolerance for over-spending on projects.

Initial budget allocation was done by management of Minerva; but user story estimation was done by developers as they were perceived to be better positioned to elaborate on complexities of work that *they* need to do. However, their estimation accuracy seemed to improve as they progressed through the project. To measure the financial health of a project, remaining story points were compared to the velocity of the project teams which provided an indication of the time required to complete. A project was considered to be in good financial health if the time remaining was projected as before the delivery date. Reporting of deviations was done by the Scrum Master leaving the developers to concentrate solely on designing and developing software. Monitoring and reporting depended heavily on software management support tools which in turn depended on the real-time updating of work status by the developers. Estimates for completion stemmed from the calculations performed by the software which in turn was based on close monitoring of user story completion rate statistics. The Scrum Master was often seen prompting the developers to timeously update status of their assigned user stories. Interventions from management were triggered when the Scrum Master reports showed lagging project schedules. At Minerva a more detailed account of costing was possible due to more access into this area compared to the Athena case study. However, overall and similar to the case at Athena, cost estimation was still in a state of evolution. The following narrative supports this claim:

> "We are agile to some extent, but from a project management tracking perspective it is very difficult; particularly interruption ratio and multiplication variable in hourly rate costing. I'm losing confidence in the ability to properly track cost of projects" (min_obs_sm_pg16:4).

6.4.4 Human resources management

An earlier subsection described staff allocation to new projects. This section deals with deployment, or more accurately redeployment, of staff to projects that appeared under pressure. Software development work is more akin to knowledge intensive work and the major resources tend to be the expertise of developers. Other resources, such as computer equipment, software technologies, and office space, were usually not competed over - unlike materials commodities in traditional manufacturing environments. Being an SMME, human resources at Minerva were limited and redeployment proved an ongoing volatile practice.

Human resources management is representative of a large field of academia and just as in the case of Athena, the Minerva case study only focused on the responsibilities of redeploying personnel that impacts directly on project management.

Sub—inst—object: FogBugz (instrument) and the daily stand-up meetings (instrument) provided the Scrum Master with information on developer project assignments. Artifacts like burndown charts (instrument), budget reports, and project velocity (instrument) served as indicators for project progress. Lagging progress served as triggers for the practice of human resource redeployment.

Sub—rule—com: From the pool available at Minerva (community), developers (subjects) who were least busy (rule) or developers that have previously worked with that type of technology or for that client (rule), were selected by the Scrum Master (subject) for redeployment to projects showing signs of distress. These redeployed developers were known as contractors.

> "The strategy is to have a technical lead and to have a pool of resources who are available to assist on any project that's under pressure at the time" (min_int_sm_pg6:2).

Lead developers sometimes asked for additional manpower for a day or two during daily stand-up meetings or by approaching the Scrum Master. This sometimes led to the Scrum Master having to lobby support before redeploying developers as contractors to other projects. The following conversation bears reference:

> "[Technical Lead 1] wants everything in this sprint because he wants completion asap" (lead developer).
>
> "Not possible, but I'll speak with him. I'll set a new milestone for the project" (Scrum Master). (Min_obs_pg7:5).

Shuffling developers between projects (rule) presented certain risks and this practice mostly represented a *journey of discovery* for management (subject) at Minerva (community). The following interview snippet with the CEO highlights these challenges:

> "There definitely are challenges. This is very interesting. When we started off, [as a] small company we would pretty much have one developer per project... And then we'll have a silo of information which was a risk. But, it was very efficient because you didn't have to skill up and get this knowledge transferred to be able to work on another project… But, when we started growing and one project would require two three developers… One way we addressed that was to say well, let's create teams and let's put the team on a project for two weeks and then the same team moves on to another project for a different client…The problem we had with that approach on the one hand is that nobody would take responsibility and ownership… The other problem was velocity. Whereas you would have one developer working on a sprint, you now have three so the velocity was too high. Too much been done too quickly to respond to change... The other problem we had was pipelining (pipelining refers to estimated, documented, and allocated) work. We need a whole

> sprint ready for three people to tackle by the end of the week… So we went back to the slower velocity. Let's rather create ownership with the developer so one person would own the project. But you try to break down the silo so if they get sick or go on leave we can still continue. We have the ideas like code review or never estimate on your own, there is always some other team member even if they are not doing development they are involved in your estimates and code reviews and architectural decisions so that you do break down that silo but you still do have that responsibility and the slow enough velocity to change, but you can pull in other developers if you need to get the speed up really quickly" (min_int_ceo_pg2:6).

There were many observed cases of developers (subject) who volunteered assistance (rule) with projects that were falling behind schedule or when they had technical expertise that would contribute towards improving a project (community). The following narrative highlighted this:

> "And if we are ever out it's just a case of people putting in the extra hours. There's quite a few people, myself included, who will put in extra time in our evenings and not even log it or anything because we just want to do more investigative work which will help benefit other projects as well as the current project. People are very committed within the company" (min_int_dev6_pg24:5).

The narrative above sheds light on the commitment of developers to the craft of software development. Further, the demonstrated attitude of being willing to help suggests that the lack of strict job boundaries may play an important role in building community in a SMME. The willingness to help is done spontaneously, suggesting the existence of a community of practice and support for one another.

Com—div of labour–object: The Scrum Master (division of labour) was primarily responsible for redeploying staff (object) and in some cases he did consult with the Technical Leads or Commercial Director (community). From the pool of developers at Minerva (community), developers were chosen as contractors (division of labour) for projects under pressure. This reassignment (object) was temporary. Contractors worked concurrently on their original projects as lead developers or as coding buddies (division of labour) and on newly assigned projects as contractors (division of labour). The Technical Lead speaks of margins:

> "We can scale up during the project. The highest number of contractors was last year (2011) when we had six on a project" (min_obs_tl1_pg7:4).

One of the propositions of Brooks Law was that adding more developers to a project behind schedule would only serve to make that project later. Was this still relevant in Scrum context?

> "I do think so. I think we had projects that demonstrated that nine women can't make a baby in one month. Certainly the more resources you put onto a project, again software development is not a manual labour process. It involves cognitive thought, it involves in some aspects a lot of creative and abstract thought. Like putting twenty philosophers into a room to debate something, doesn't mean that you will come to a conclusion faster. In many instances you will add more confusion and raise more questions. And I believe the same holds true for software development in that by

adding more cognitive thinkers into a project, each with their own style of thinking, their own way of thinking" (min_int_sm_pg10:3).

The interview question that immediately followed the above interview response, sought clarification on the observed practice of moving developers or contractors to lagging projects. The developers added as contractors (division of labour) were usually given stand-alone user stories to code (object); that is, software components which did not integrate nor had little dependence with the rest of the system. Further, developers who have worked on a project before (community) will be re-allocated to the same project if there is new work to be done (division of labour).

Ripple effects were the major cause of concern with the practice of redeploying developers. These included maintaining a balanced velocity across projects and ensuring adequate agents for knowledge transfer within the project. Contractors were added to projects to maintain the projected velocity. There was never a case were more than necessary reinforcements were deployed to a project resulting in an unwanted effect of too high a velocity. The lead developer was usually the one that worked on the same project from beginning to end. Brooks Law still held in this context and the number of contractors added to a project were carefully considered and kept at an optimum. Developers sometimes put in extra hours to achieve milestones. Developers also volunteered to help even though they did not have spare time. Redeployment of developers was not a completely seamless practice with instances of both Scrum Master and lead developer needing to lobby for additional manpower within a project. This showed a shared accountability for the project schedule by both the developers and the Scrum Master.

6.4.5 Infrastructure design

Infrastructure refers to the items that were required for the process of software development. These items included renting office space, purchasing desks and chairs, and acquiring technologies such as software, compilers, project management software, communications, and hardware, and ICTs. However, fieldwork at Minerva revealed that infrastructure was not competed over.

Sub—intr—object: There were no observed instances of this relationship. However, developers (subject) did inform the researcher that requests for items (object) such as new office furniture was done informally (instrument). The implementation of new development

software such as compilers and ICTs (object) were usually discussed during technical sessions (instrument).

Com—div of labour–object: The partner owners were responsible (division of labour) for maintaining an environment suitable for Scrum software development (object).

> "My title is a Technical Director. My core responsibilities is to ensure that the team that we have here are operating with the right tools and in some respect the right processes and have an environment that's conducive to the type of work that we do" (min_int_tl1_pg4:1).

Observed examples of this were the three open plan offices for the developers and the boardrooms available for meetings and discussions. Developers could make requests (division of labour) for additional or newer infrastructure (object).

Sub—rule—com: In keeping with Scrum guidelines, developers (subject) shared an open plan office (rule). Senior developers (subjects) were positioned two or three to an office while the Technical Leads, Commercial Director, and Scrum Master (subjects) had separate offices (rule). Daily stand-up and other forms of meetings occurred in the boardroom (rule). Quiet time (rule), or sometimes called *boom time*, was observed for four hours in the morning until lunch and then again for three hours in the afternoon. During this time disturbances, such as cell phone conversations, were not permitted in the open plan offices.

Infrastructure acquisition and maintenance did not surface as a complicated string of relationships. This was due to the complete lack of competition for physical, technological, and communications based resources. Minerva had more office space than it required at the time of fieldwork. The quiet time rule was deemed a necessary mechanism to reduce chatter in an open plan office in which knowledge intensive work in taking place. This went against the principle of osmotic sounds; that is, learning from background chatter. However, production was perceived as improved with this approach. Developers could request items such as office equipment informally. However, changes to development platforms and the purchase or download of development software was usually subjected to deliberation in a meeting.

6.4.6 Progress monitoring and reporting

The practices within this knowledge area attempted to provide both teams and management with accurate daily project status reports. The Scrum Master was the focal actor with chained relationships to both developers and management. Observation notes revealed:

> [Scrum Master] spends a lot of time phoning lead devs working on the various projects, following up on progress, following up on suspected erroneous entries in FogBugz, and acting as liaison between devs and client queries. Each client request is handled as a new case (min_obs_pg8:5).

The progression of active projects was constantly monitored through both formal and informal practices. FogBugz was an important ally in the monitoring activity. In turn, this practice facilitated adjustments to projects; for example, adding more manpower or renegotiating client requirements. The primary unit of analysis was user story points.

Sub—inst—obj: This was mostly an activity for the Scrum Master (subject) but he relied heavily on the reporting features of FogBugz, TimeTracker (instruments) and the developer's (subjects) commitment to timeously updating project data on these instruments. The following set of snippets from observation notes reveals the importance of the instrument:

> Fogbugz is a case tracking tool. First, assign a case to a dev. Each developer gives their own time estimate. Fogbugz learns about the developer performance based on heuristics. This is how [Scrum Master] and the Product Owner can track progress and estimate time for sprints (min_obs_pg3:3).

> There is a separate application called Timetracker that interfaces with Fogbugz. This is in-house developed software that interfaces with Fogbugz. Fogbugz has got a built in heuristic algorithm that calculates the projected time for completing a case, per developer (min_obs_pg5:1).

> The time tracker interface also has 2 time indicators. The green time is the time the developer starts work, this he/she enters. Timetracker calculates and displays the time spent thus far in red. These times are stored in Timetracker's database server and is maintained for each dev. Fogbugz would then use this database to create a historic performance measurement for each dev (min_obs_pg10:3).

Primary supporting artifacts were the burndown and velocity charts produced by FogBugz (instrument), as well as a budget expenditure spreadsheet (instruments) maintained by the Scrum Master (subject). The burndown charts illustrated the predicted time remaining for a sprint, and thus provided an indication of overall progress. The velocity chart tracked the user story points completed at a given point in time against the total number of user stories for that sprint. This provided a real-time completion rate of a given project. The budget expenditure spreadsheet provided a holistic view of the budget consumed. Using these project artifacts, the Scrum Master (division of labour) could track progress of projects and report on any *red flags* (object).

Sub—rule—com: There were two major observed project types at Minerva (community) which were first introduced earlier. These were fixed cost and fixed scope, and time and materials based projects. Progress reporting varied across these two project types.

Assuming a single sprint, in a fixed scope and cost project (community), it was during sprint planning that the user stories for that sprint were captured in FogBugz in the form of a case

list (rule) by the assigned developers (subjects) during sprint planning. User stories had an associated estimated effort and priority. (More on this is provided in a later subsection which focuses on planning). When a developer began work on a user story, he or she changed the status of that user story to *active* in FogBugz (rule). User story states were either active, parked, in progress, resolved or pending code review. As user stories changed state to *resolved*, this indicated completion and FogBugz (subject) registered certain parameters including this state change, the time taken to complete, and the responsible developer (rule). Actual time taken to complete was compared to the estimated effort for that user story. Using built in algorithms, FogBugz estimated the amount of time remaining for the sprint (burndown) as well as the speed at which user stories were being completed (velocity). The Scrum Master (subject) reported on this progress during the daily stand-up meetings (rule). Based on these reports, decisions were made; for example, in one observation the Technical Lead (subject) suggested adding contractors (rule) to a project, while in another the Scrum Master (subject) made the decision to *park* certain project work (rule). In essence, the sprint should be completed within the time frame such that it remained profitable, an important consideration due to the inability of the SMME to absorb continuous failed projects.

For time and material based projects (rule), the procedure for generating and capturing user stories was the same as above. However, the client was charged a flat hourly rate (rule) for completed work on the project. As user stories in FogBugz underwent state change, for example, from in progress to resolved, the Scrum Master (subject) reported on work completed and work remaining.

For both categories of project (community) progress visibility was paramount (rule):

> "Making project progress visible is the first stage of corrective action" (min_obs_sm_pg13:2).

The Scrum Master commented further on the budget expenditure spreadsheet that he maintained that:

> "This provides a rough estimate so that the team knows where to spend their effort in the upcoming month" (min_obs_sm_pg14:4).

A developer was asked about the procedure when a project went over budget:

> "If we are ten percent over budget, it goes red…We are called in for a meeting and they [management] tell us we are over budget and we need to sort ourselves out. If we need to work overtime we can just so the project can finish on time" (min_int_dev4_pg17:3).

Com—div of labour—obj: Within the context of this SMME (community) the Scrum Master had the responsibility of monitoring and reporting progress (object) of all projects. This was a

first step towards his primary function of removing impediments for the developers and designating developers as contractors (division of labour). If imminent anomalies surfaced within any of the projects these were immediately reported (division of labour) to management who then made executive decisions regarding that particular project (object). Developers (division of labour) were responsible for real-time updating of user story status (object) within the project (community). Developers (division of labour) needed to view progress at any point (object) and the Scrum Master was quizzed on this during the interview:

> "That's a good question. They could conceivably go into the schedule view in FogBugz and look at the burndown chart and anticipate the time remaining on ebs. So that isn't restricted to admin users, anyone could view that. Whether they do it or not is debatable. I guess there's a large dependency on the information that's presented during the daily stand-ups and they execute based on that information" (min_int_sm_pg7:1).

A developer added:

> "Currently the only system we have is the morning stand-ups and the excel spreadsheet that [Scrum Master] uses showing percentage of budget used and percentage of completion. If there's a discrepancy he'll flag it as a warning sign… It's a new system we've been using for a few months. I'd say it's working well" (min_int_dev1_pg11:2).

These narratives stress the prototype nature of the monitoring and reporting activities at Minerva. The ability to experiment with alternative program management practices may be easier, but at the same time are crucial for a SMME when compared to large corporates.

Fieldwork showcased the Scrum Master (division of labour) as the central actor for progress monitoring and reporting practices (object). There was a strong dependency on project management software, namely FogBugz and Timetracker. Reporting of progress was mostly in real-time, with the Scrum Master performing at least a daily report on progress. Developers could view burndown and velocity charts at any point as these were constantly updated by FogBugz. However, accuracy required on-time updating of user story status by developers. Based on these reports, management did intervene by moving human resources around or renegotiating requirements with the client. Real-time reporting of progress allowed for *on-the-fly* adjustments to projects to bring them within acceptable range. Sophisticated prediction models built into FogBugz and Timetracker were essential for monitoring projects. Fieldwork at Minerva allowed for detailed observation of two projects types and their subtle differences in reporting mechanisms. It must be noted that the monitoring and reporting mechanisms were in a state of probation at the time of fieldwork but personnel seemed confident in its operation. This narrative by the Technical Director makes a case for this:

> "Our process has changed quite a bit in the last two years. One the reasons why it's changed quite quickly, and I think change is the wrong word instead I'd say evolved, we've got the metrics to measure how things are impacting our business. So because we log time, because we estimate on work it's very easy to see when things aren't working" (min_int_tl1_p4:2).

6.4.7 Planning

Planning activities at the program management level primarily involved review and reconciling of individual project plans, with a subsequent holistic image of planned tasks, schedule, and milestones. The delivery date for a project was set by management while the lead developer and his or her coding buddy were responsible for planning work carried out within the scheduled margins of that project. Planned project activities were noted on FogBugz which then became publicly accessible.

Sub—inst—obj: The lead developer and coding buddy (subject) were responsible for devising and carrying out individual project plans. Individual project planning entailed the setting of effort estimates for the user stories, prioritization of user stories, and then capturing these as case lists in FogBugz (instrument). This prioritized list formed the product backlog (instrument). FogBugz (instrument) provided predictions of completion dates for projects based on the user story effort estimations (object) on its database of statistics of the developer's completion speeds. The Scrum Master (subject) viewed these predictions and then derived milestones and suggested progression rates for each project, using the metric of user story points.

Sub—rule—com: For predictive calculations, FogBugz worked with a probability setting that the Scrum Master (subject) usually set at 50% (rule). Probability was required to derive a range; for example, a delivery time best case scenario was 120 hours while worst case scenario was 180 hours. Probabilities were set in order to provide more realistic milestones. More realistic because certain anomalies could manifest within the different projects at Minerva (community) such technical debt (rule), interruption rate (rule), and "unbillable broken windows" (min_obs_sm_pg13:2). Technical debt was described as estimating the amount of rework or corrective work that occurred in a software project. An example of interruption was when developers (subject) took sick leave.

> "FogBugz learns interruption. E.g. if a developer states 1 hour to complete a case in FogBugz, but he actually takes 1.5 hours this is an interruption. Also, if a developer is sick for an average of 2 days a month, it learns this" (min_obs_dev5_pg12:3).

Another anomaly was unbillable broken windows which referred to user stories that should not have been assigned effort estimation during project planning. Observed examples

included corrective work from a previous sprint and simple maintenance tasks that literally required five minutes of work.

Work had to be planned before tasks could be assigned. This was often referred to as pipelined work. There were instances where developers had a waiting state as indicated in the following:

> One developer comments that he needs more cases. To which [Scrum Master] replies:
>
> "Check with [Technical Lead] to sort out cases and then move into backlog" (Scrum Master)
>
> (Min_obs_pg14:3).

Projects that were heading over budget or beyond schedule also triggered planning or re-planning by management (subject). Either more developers were added (rule) or prioritization of user stories was renegotiated (rule) with the client so that some could be removed from the sprint (community). When asked about client reaction to renegotiation, the typical answer is personified in the following statement:

> "I can't recall that we've had a bad experience" (min_int_dev1_pg11:3).

Provided that clients had trust in the methodology, they seemed to more easily comprehend this natural process of on-the-fly planning and necessary renegotiation.

Com—div of labour—obj: Developers (division of labour) took care of planning (object) work at the project level. This observation snippet provides some insight:

> The project starts with planning. [Developer 3] and [developer 9] are the Product Owners at this stage. They will act as liaison between [Minerva] and [Client]. [Developer 3] and [developer 9] will meet later today to do planning. They will develop high level use cases. These will then be broken down into more technical cases concerning UI, Facebook interface, DB changes etc. During this process questions will arise for customer and the team. [Developer 3] and [developer 9] will then create an enumerated backlog of features. No story points but t-shirt sizes are used to represent complexity. The management team found that numbers where being assigned based on the time (in hours) needed to complete the story. This backlog is then broken into sprint by the Product Owner and the technical lead (min_obs_pg3:3).

The Product Owner played a supporting role in planning (object) by helping with prioritizing user stories in the product backlog (division of labour) for a project (community). However, this is not completely devoid of challenges:

> "Agile works when Product Owner and Business owner are very close together because then we can negotiate and control deliverables. Agile becomes difficult for [an] external customer. If there are bug fixes or forgotten requirements, and we have already moved on to the next client, then this can be problematic" (min_obs_sm_pg4:1).

The Scrum Master (division of labour) had the responsibility of monitoring and reporting progression (object) of projects (community) based on these plans.

Once the development team had been established for the project (community), the developers selected their own user stories for development (division of labour). As a result, there were some instances when this plan was disrupted; such as when the Scrum Master forced work plans:

> "The following work has to finish today" (min_obs_sm_pg8:4),

And a developer was asked to act as a contractor:

> "Do you want to be pulled into the current sprint?" (min_obs_sm_pg8:5).

Although planning practices seemed on the surface underpinned by a principle of self-management, its application appeared somewhat cautious:

> "We have [had] this debate often in the last couple of years that a lot about what we read about these self-managing teams in the US doesn't work in South Africa and that is largely attributed to different personality types in South Africa compared to the US. For example, here people generally avoid conflict. It's just a cultural thing in South Africa you just don't want to offend people. In a team environment if someone is clearly wrong people might not stand-up and voice their opinion, and just go with the flow to avoid conflict. Whereas, I think in different countries, you might get different cultures that are much more open and they voice their opinions. That's where flat structure, self-management teams might work better because there's no cultural barriers to proper open collaboration. It's just a theory that we have while you hear about these self-managing teams working so well while we have really smart people working here but we have these challenges where we still do require someone to oversee them and make the tough decisions" (min_int_ceo_pg1:3).

The initial budget and delivery dates were established for each project via negotiation between the client, CEO, and Directors. Planning at project level was done entirely by the developers but within the margins of the delivery date. Their planning compensated for unknowns such as first encounters with new technologies or new areas of computing. That said, there were no detected checks in place to ensure that the developers were not over estimating user stories and as a result buying themselves more time. Instead, there appeared to be a strong sense of trust between developers and management. This was facilitated by SMME small workforce and strong working relationships. Management did trump the developer's project plans when the need arose; but, given the lightweight planning, their interruptions appeared to be easily absorbed. Clients were generally open and willing to renegotiate priorities of user stories provided they understood this to be a natural activity within the Scrum environment. (More on this in the next section). One glaring observation is the high probability setting of 50% for planning. Overall planning was deemed an important

practice, highlighted in the following narratives, which were responses to a question on quality characteristics:

> "Accuracy in terms of estimation which is a big part of quality which often gets overlooked. Because a quality product starts with quality planning. If you haven't planned correctly and haven't been able to predict the time periods and the interdependencies of the cases [user stories] then your project is going to have difficulty succeeding" (min_int_dev9_pg34:2).

> "Planning is number one. We've fallen down so many times on planning. We failed repeatedly just on the great wall of unknowns. So many times we know we budget 100 hours for example but we send 150 hours because every single time we just get hit with things we just didn't know about. Being kind of cutting edge we are permanently implementing new solutions and they often involve platforms we know nothing about… The more successful projects that we've had was when there was a lot more planning upfront. Unfortunately, that is not something that we have the luxury of" (min_int_dev3_pg13:4).

6.4.8 Client liaison

Reporting project progress, clarifying requirements for developers, renegotiating client requirements and client requests, are some of the practices that fell within the portfolio of client liaison. Ideally, a Scrum software project requires a client liaison situated in close proximity to the development team. However, given the demands of modern day business, clients do not always dedicate manpower for the purpose of client liaison. Also, for well understood requirements associated with simple software systems and recurrent clients, a client liaison is not always feasible. In the absence of the client liaison the Scrum Master provided a channel for communication with the client. Apart from communication of economic aspects, client liaison also embodied practices that served to preach and convey the practices of Scrum to clients unfamiliar with this approach.

Sub—instr—obj: Client liaison (object) was normally the responsibility of the Scrum Master (subject) who utilized a plethora of artifacts. Project schedule and budget information were primarily derived from the project management software FogBugz (instrument) while project artifacts such as burndown charts, velocity charts, and the budget spreadsheet (instruments) provided feedback whenever the client requested it (object). Sometimes the Scrum Master (subject) sought clarification via quick, informal conversation (instrument) with the developer in charge before responding to the client request (object).

Software tools such as Base Camp served as reporting and communication interfaces between development teams and between development teams and clients. The Scrum Master shed some light on the value of Base Camp software:

> "The problem with any tool is that it's only as effective as the people who use it. I think we always had this challenge in trying to convince, and it's not just my time spent here at Minerva but even in

> my previous place of employment, customers are lazy. You know, even if you try to explain the value of using a particular tool or technique or process, human nature is always to take the path of least resistance even if it is not for the inherent good for the project or whatever the case may be. So, certainly for the purposes of internal collaboration, I think it is a very useful tool. I think it is a good place for discussion or knowledge sharing and a place of reference if you wanting to find specific information about a project. But in context of discussion with clients, again subject to their willingness to engage and actively use it. Some people are very good and some people will revert to using e-mail; unfortunately e-mail is the wrong medium to use for collaboration" (min_int_sm_pg8:2).

Seeking clarity on requirements (object) and on ambiguous client requests (object) was done via a host of ICTs such as telephone or e-mail (instruments), or via formal meeting requests (instruments) with the Product Owner (subject). Renegotiation of client requirements (object) usually took on the form of user story prioritization, which subsequently involved updating the product backlog (instrument). This was done as a formal meeting (instrument) typically attended by the lead developers, Technical Lead, Scrum Master, and Product Owner (subjects).

Sub—rule—com: The Scrum Master (subject) was charged with dealing with client queries regarding project budget and schedule (rule). The Scrum Master's (subject) portfolio also included removing impediments for the developers (rule); hence he sometimes sought information from the client (subject) in an attempt to alleviate difficulties which might have arisen within any of the different projects (community).

Calculation and reporting of schedule and budget usage (rule) was done by the Scrum Master (subject) which left the developers and the Directors free to concentrate on their primary task of software development (rule). Reporting of scheduling looked as if it was heavily dependent on developers updating user story status timeously (rule):

> [Scrum Master] spends a lot of time phoning lead developers working on the various projects, following up on progress, checking for any erroneous entries in FogBugz, and acting as liaison between developers and client queries. Each client request is handled with a case. For example, after receiving a request from a client, he contacts the lead developer and discusses when to include the request in the sprint. He also confers with Technical Lead on this one before making the decision:
>
> "I'll tell the client that it will go into the next sprint." (Scrum Master) (min_obs_pg8:4).

From the onset of most projects (community), Minerva (subject) actively engaged in a type of agile induction for clients and educated them (rule) on the Scrum management framework:

> "From the beginning, the first phone call we talk about agile and our approach. You would have seen on our website that we have quite a bit of information about our agile approach. In our proposals we include an overview of the approach we take. And I know that with some clients we have actually done like little training and little presentations about agile and why it works. So we

> really try to get buy in. But at the end of the day we realized that you just need to demonstrate it. We need to run a small project, show them how it works, build up that level of trust and then the relationship evolves from there" (min_int_ceo_pg2:4).

The Director stated that this was sometimes a difficult exercise and related a past experience were it took almost nine months before a big client finally felt confident enough to award a software contract to Minerva.

Although the Scrum Master (subject) was the primary contact between client and development team (subject), a recent practice saw the inclusion of the developers in formal client meetings (rule):

> "There are three groups of people: the clients, project management, and the developers. So essentially those three people need to be on the same page. Because there is a middle man i.e. the manager between the client and developer. Which is why recently there was a decision to include a developer in all client meetings. That wasn't done in the past but I think it's very useful to lower that risk" (min_int_dev8_pg32:4).

Com—div of labour—object: It has been established that the Scrum Master (division of labour) is the hub of this activity, the point of contact (object) between the client and the project team (community). Queries to and from both the client and the development team where channelled via him. Reporting of project progress to the client (object) and of budget usage for projects to the development teams and to management (object) was the Scrum Master's responsibility (division of labour). Renegotiating client requirements occurred when there was a client change to functionality or when a user story was miscommunicated or misunderstood. Renegotiation of requirements (object) with the client was usually done by the Directors, the Scrum Master, and sometimes the developers (division of labour) from the project (community). The importance of developer inclusion is highlighted below:

> "Depends on the type of meeting. If it has to do with any technical aspects of the system then it's vital that the developer sits in because often decisions are made and the developer might not understand why" (min_int_dev7_pg29:4).

Additional functionality requests (object) for the different projects (community) were made via the office of the Scrum Master (division of labour). Although maintaining a healthy client relationship was a strategy of SMME, it was not entirely without boundaries:

> Client sends [Scrum Master] a support request. The client wants it done today. [Scrum Master] then telephones the developer and tells him to refer any communication from the client to him and that under no circumstances must current work [today's work] be disturbed (min_obs_pg10:2).

"Support" refers to change requests or changes to functionality to software that had already been released. Clients were encouraged to e-mail their list of support items so that they could be converted to user stories or be assigned to the lead developer for uncomplicated requests.

Educating Minerva's clients (community) in the Scrum approach (object) and in reasons for its adoption (object) was the job of Directors (division of labour). However, all developers were acutely aware of the importance of client liaison as demonstrated by this interview response:

> "Definitely important. That's one of the things we try and cultivate from when they first meet the client right through to building a long standing relationship with them. Because a lot of the time the clients that come to us are expecting us to drive the processes. You have the smaller companies that don't really interact with the client as such, they sort of have a list of demands that they want, what's the best price for it, and by when it must be done. Whereas at Minerva, they come to Minerva because they know there's a trusting relationship, it's there to be built. We have our reputation within industry and a lot of clients that are dealing with us now have come to use through that reputation" (min_int_dev6_pg25:5).

Client liaison was an area that posited the Scrum Master as the primary actor. At Minerva, he was the single point of contact between the development teams and their respective clients. In order for him to provide accurate reporting to clients and to teams, project management software and the accompanying self-maintained budget spreadsheets, were artifacts of paramount importance. Seeking clarity from clients was channelled through the Scrum Master whenever the team did not have the luxury of full time access to a client liaison situated among the team. Since renegotiation of requirements affected the profit margin, this activity was done by the Scrum Master and the Directors. Frequent client queries were not frowned upon; instead, client queries were deemed an important attribute in client relations and nurturing the client's sense of involvement. Seeing as Scrum was heralded as not yet mainstream in the South African software development context, client education of the process and approach was vital. This equipped most clients with the necessary knowledge of Scrum management to allow for easier integration with their existing management processes. Being a SMME without a separate marketing division, getting client buy-in of both the software product and the Scrum process was a necessary first manoeuvre for dealing with most new clients.

6.4.9 Project schedule risk identification and mitigation

There are many risks associated with software project management. As was the case at Athena, the fieldwork at Minerva focused on project schedule risk. These risks mainly included overspending of budget, missing schedules, and loss of resources. This section looks at the practices that were mobilized in identifying and seeking mitigation of these risks.

Sub—instr—obj: "Making project progress visible is the first stage of corrective action" (min_obs_sm_pg13:2). Observed Scrum practices - for instance, daily stand-up meetings

(instrument) in conjunction with support tools such as FogBugz, made developers, the Scrum Master and senior personnel (subjects) aware of flaying projects (object). Informal communication (instrument) between the Scrum Master and the developers (subjects) with regards to imminent shortfalls in project progress (object) were also noted. The budget spreadsheet (instrument) maintained by the Scrum Master (subject), helped to create financial health awareness (object) of the program management level which was communicated to the entire workforce during the daily stand-up meetings. No visual aids were utilized.

Sub—rule—com: Developers (subject) assigned to a project (community) performed the user story effort estimations (rule). Estimation of user stories was done as a team effort (rule) to further reduce the risk of inaccurate schedules within projects (community). This is a form of risk mitigation since it encouraged the consideration of different perspectives on estimates. Apart from this schedule related risk, campaigning in new fronts such as unknown business domains or implementation of new technologies also represented risk:

> "We are working with real people. When a dev is not as experienced or there is new tech then we move more seniors into the project and accept the cost of that" (min_obs_sm_pg13:6).

In the case of a developer (subject) working in unfamiliar areas for the first time, the coding buddy was usually a senior personnel member (rule), like a Technical Lead (subject) or someone who had experience with that technology or in that domain. Furthermore, the effort estimations were inflated to compensate for the additional effort that would be expended on learning when no other personnel within the SMME had prior experience or expertise:

> "The risk of new tech or new domains is added into the project through t-shirt sizes [estimation metric] during planning. Here the devs themselves evaluate the risk. Incorrect planning i.e. t-shirt estimate, might occur but that's why it is a team effort to help reduce this occurrence" (min_obs_dev2_pg13:4).

When a project (community) velocity slowed and exhibited the risk of missed deadlines, the simple mitigation of adding more resources (rule) did not always suffice. The Scrum Master explained:

> "Well I think it depends on why the velocity has slowed down. In every instance there can be an intervention. That intervention depends for example, if the velocity has slowed down due to customer feedback being slow, the only intervention one can take is to escalate and inform the client, listen your lack of response is impeding our ability to deliver. This is the impact in terms of time and potentially this is the impact in terms of cost… there's always going to be unknown elements when customer engagement is required because you can't control that. And there are instances when we need to push back on the customer and say if we don't receive feedback within a certain amount of time it will mean that we need to shift resources onto another project and we can only recommence with your project once we receive feedback" (min_int_sm_pg10:1).

With the unforeseen manifestation of additional cost and later delivery (rule), negotiation with the client (subject) ensued, led by management and the Scrum Master (subjects). However, the question of reasons for the need for negotiation surfaced. The following narratives explore this:

> "It's a tough one. Because you don't really know. It's always unknown. What I find myself doing now is when I'm estimating cases I'm padding them up massively which is also a negative thing because the future clients are getting billed potentially for something that did end up taking one hour. So it's a very hard thing to do. Like I said, we are trying to implement proof of concept sprints where the customer pays up front and we say we'll spend 16 hours looking into this because we don't know how it works. But a lot of clients are not happy with that, they want an upfront project estimate. That's an area we are trying to improve on but I don't see any ways around it yet" (min_int_dev3_pg15:1).

> "I think there are always areas in a project that have inherit complexity, inherit risk especially with estimation. So we can often, depending on when the estimation is done, if it's done quite far in advance and as we working on the project sometimes the work that we estimate is only our estimates and don't truly reflect complexity at the time of planning. You didn't realize all [the] complexity. I guess the greyest area would be when estimates were done without the greatest level of certainty" (min_int_dev7_pg28:2).

The general rule of thumb evident from observation seemed to be that if the slowdown was caused by internal (community) reasons, then explanations were sought (rule). If the slowdown was due to technical difficulties, then a more senior developer (subject) was deployed to assist in coding and coaching.

Loss of resources referred to the practice of developers being decommissioned from one project and assigned to another to improve productivity, a necessity given the limited resources available to the SMME. Developers (subjects) assigned to projects that were ahead of schedule or assigned on more long-term time- and material-based projects, were redeployed (rule) as contractors to lagging projects (community). This practice presented risk. The Technical Lead elaborated on some experienced complications:

> "It's not ideal. We've got the challenge and it's a legacy challenge. As we grow the projects we take on are bigger, the cycles are longer, you can get more people involved in the project and it's realistic to do so… Over time we've accumulated a lot of clients and a lot of different systems. We feel guilty about saying to the client sorry, we not going to help you anymore… If we could split the entire workforce down the middle and say you working on project x and you working on project y for the next year, that would be the ideal environment. But that's not the situation we have and there's a fair amount of shuffling. We mitigate those issues by getting other developers involved in code reviews at least. I'm not going to pretend that is the best solution but it does go a long way" (min_int_tl1_pg4:6).

In order to cater for developers taking leave or getting sick, the coding buddy (rule) was an approach towards mitigation (object). Other risks that may indirectly affect project schedule were described by this senior developer:

> "The risk is that when you get into the nitty gritty of the project and you've stopped the creative process. Because when you first start you have high ambitions you want to create a Rolls Royce product and once you start getting into the challenges you lose the creative edge. And so, when you are in the creative zone you really want to max out and the risk is that some parties become too controlling, whether is management, customer or developers and the creative spirit is killed off" (min_int_dev1_pg11:4).

> "The risk of over deliberation. Especially in an idealistic environment. Understand that if you are in a very profit-driven environment and every single action is being measured against profit. And the risk of becoming too idealistic and too perfectionist is that you actually start to work on material that has no financial gain. So that's a heavy area and it causes projects to run late, it causes clients to be billed for more than they had planned for and you run the risk of making the relationship go sour" (min_int_dev1_pg11:5).

Com—div of labour—obj: Risk identification and mitigation were not practices segregated to a particular grouping of personnel. Developers (division of labour) were responsible for recognizing and mitigation of risk on a day-to-day basis (object) for their allocated projects (community). They also employed techniques like the coding buddy system and teaming up with seniors as a form of risk mitigation. The Scrum Master reported (object) on risk with regards to project schedule (division of labour) and their respective budgets (division of labour). Developer allocation awareness also featured in this reporting which in turn facilitated reallocation of resources to projects at risk. This reallocation was usually a result of deliberation (object) by the Scrum Master and Directors (division of labour). The risk of the Scrum Master being on leave could be sustained for a short period of time since project work is disseminated within the self-organizing group. Other senior personnel could take care of multi-project allocations due to small workforce and informal communication which allows for project awareness.

Risk identification was in real-time. With software like FogBugz, practices like daily stand-up meetings, and a small, close knit community of developers, reports of projects at risk were quickly spread among the ranks of the SMME. This allowed for a quick reaction and no project was observed getting into serious trouble. Risk mitigation was not a task without adverse ripple effects, especially in the instances where the client was unwilling to allocate more time or did not provide timeous feedback. In those cases clients were urged to cooperate. Developers were usually deployed in the project at risk sometimes causing a drawn ripple effect. Risk in the form of overshooting schedule and over expenditure of budget revolved around the accurate representation of project user story status. Indirect risks were too much effort expended on perfectionism and on extensive deliberation as well as loss of creativity. The practices of estimation were detailed in an earlier section. However, this

area proved one in need of refinement and evolution and thus it has been situated under risk and mitigation.

6.4.10 Aligning practices with business strategy

Business strategy represents a large field of academia and perhaps an entire thesis worth of writing would not do justice to the subject matter. Similar to the Athena case, many different business strategies were in place but this thesis focused on the program management practices in support of strategies of self-managed teams, client interaction and involvement, and teaching and coaching development teams. Practices in this knowledge area sought to ensure that the business ethos manifested in each of the projects undertaken at Minerva. Accompanying practices also sought to provide a feedback loop in order to evaluate the business strategy.

Subj—instr—obj: Scrum practices like daily stand-ups (instrument) were vital in maintaining the strategy of self-managed teams (object) because they provided a mechanism whereby project awareness was maintained by developers (subject). Product backlog creation and evolution (instrument) and sprint planning (instrument) were practices that also emancipated project teams (object).

Daily stand-up meetings (instrument) doubled, to some extent, as a tool for teaching and coaching (object) development teams (subject). However, the primary practices in sustaining this strategy were the coding buddy (instrument), technical workshops (instrument), and the monthly developer meeting (instrument). Informal communication (instrument) was also observed as advancing the practice of teaching and coaching (object). Technical support for teaching and coaching (object) in the form of an internally maintained wiki called Yammer (instrument) served as a central library and online chat medium.

Client interaction and involvement (object) occurred mostly while crafting the product backlog, during a subsequent grooming phase, and during the sprint review (instrument). On a day-to-day basis, the client's engagement was maintained via a variety of mediums including teleconferences, e-mail, and site visits (instruments).

Sprint retrospectives (instrument) were cornerstones to the process of evaluating business strategy and shaping operations (object). The development team, Scrum Master, Technical Director, and CEO (subjects) actively participated in these techniques.

Subj—rule—com: In maintaining the strategy of self-managed teams, Minerva (community) resembled a flat hierarchical structure (rule), typical of SMME characteristics. The Commercial Director elaborates:

> "Personally I'm very against structure. I understand the requirement for a formal structure especially as you grow. But I see within ourselves and within our clients that processes and structures can get in the way of quality and efficiency. So there is a trend for us. So I have this tension in myself. We need to grow in order to take on bigger projects but when we grow we need to put new processes in place and it becomes more expensive to develop software. So if anything, I'm very adverse to hierarchy and we try to empower our development team to keep a relatively flat structure" (min_int_ceo_pg1:3).

> "I think you need to be structured. There needs to be some type of process. But, when I don't like process is when it prescribes enough for people to stop thinking. I'm always wary of putting together a 20 page process document for x and people get into this mind-set that if I execute this process I end up with a good result. That's not necessarily true… You want to put together a process that's quite, loose is the wrong word, but quite high level in a sense that it is philosophy rather than a strict you must do steps 1 to 5. So we do lean on our team to think for themselves" (min_int_tl1_pg4:3).

> "Code reviews and developer meetings serve as a regulatory function to make sure everyone has the same standards because developers are lazy. They'll take short cuts if they need to. While developers are motivated for the most part, you just need the safety catches" (min_int_dev1_pg11:4).

The Technical Director related a case in which a developer once resigned his employment with a software development SMME because of the strict processes that were in place which did not suit his personality.

> "Driven individuals … are ultimately creative and every software developer has some level of creativity because it is about design not just implementation" (min_int_tl1_pg4:3).

That developer joined Minerva just before this fieldwork.

Interviews revealed that Scrum proved useful in supporting self-managing teams with regards to sharing accountability (rule) for the project (community):

> "It also allows for everyone in the team to share accountability on the state of the project. Instead of when the project fails and everyone says they didn't know there was x amount of budget or we didn't know we were behind the trend of where we should have been. So I think the accountability aspect certainly assists in the management role" (min_int_sm_pg9:3).

> "The teams try to find the high level goal that the customer is trying to achieve. We might document some technical aspects, but we won't mandate the how. This would then land on the team's desk. They would then self-organize, do I look up, do I need to do research on a tool that could potentially solve this problem, do I ask my colleagues, or do I develop from scratch. From that perspective I think they have a lot of responsibility on their shoulders. They are self-driven in that sense" (min_int_tl1_pg5:2).

For the strategy of preserving good client relationships via interaction and involvement (rule), some of the practices bear close resemblance to requirements for management (subject) by Denning (2010b) detailed in Chapter Two, including the notion of client delight.

> "We put everything in and we are proud of everything we do. We are very loyal to our clients and we cannot just drop them one day. Our strategy has always been to empower our clients to run their own systems and to use technologies that are common in South Africa so they can find assistance themselves if they need to. And also through the perspectives of the devs, we are learning new industries and meeting new clients" (min_int_ceo_pg1:4).

> "We don't want to be [a] company that just delivers the software product at the end of the day. We like to include the client throughout the process and they have to understand the value of the process and enjoy the engagement and understand why we chose agile. That's a lot of hard work especially for someone who is more used to a more traditional model… So I think it is important and it is difficult but if you can get a client to buy into your process and meet regularly I think it is a good thing" (min_int_ceo_pg2:3).

But maintaining good client relationships practices were a little more involved than just client satisfaction. They were partly accountable as well for project outcomes:

> "When we reach the end of the sprint, the client shouldn't go *what is this*? They must share accountability for what is being delivered" (min_int_sm_pg10:2).

Interviews with management revealed that Minerva also had the strategy of being selective of clients. They walked away from clients that did not agree with their Scrum derived practices and level of client involvement. This was stressed in the following statement:

> "Software is about people and people are different. That's the paradigm we dealing with. It's not engineering in the sense that you do steps 1, 2 and 3 and you get quality software on time and within budget. Incorrect, it never works like that. And I know that when a customer can get to a point where they can understand the constraints and the challenges then you got a customer that you are going to develop a great working relationship with" (min_int_tl1_pg5:2).

No single person is in charge (rule) of enforcing business strategy. Instead, for all personnel (subjects) business strategies are intrinsic to their daily practices. The Technical Lead elaborates on the strategy of self-organizing at the project level:

> "Ultimately if you remove responsibility from the team and take someone else in the hierarchy, and he says I'm going to tell you what to do, I'm going to make all the decisions, you lose that ownership at the developer level. They'll work from 9 to 5 and not think about the project on their way home or at night. Whereas if they have responsibility and they know that they have used component x, they will research and look for newer versions of component x and suggest using it in the next sprint. If no one is doing that, who is? If no one is actively thinking about the project, who is?" (min_int_tl1_pg5:2).

Com—div of labour—obj: Crafting sustainable business strategy was the preserve of the Commercial Director (division of labour) and the Directors (division of labour) at Minerva (community). However, when asked about business strategy, the CEO responded as follows:

> "At a high level what we looking to do is cover our overheads with new working products as revenue not consulting. Because as a business, that has been our strategy from the beginning and as I said earlier that consulting is really a means to an end. So where we like to get to is a point where product sales which are not reliant on the intellectual property rights which we are selling and the licencing which is covering our overheads. So we can have the whole team and we can talk about where we want to be spending our time not continually selling to consulting clients. This is a contrast to a vision of building a massive consulting company. We are not interested in that. We have spoken about it and it is definitely not what we are interested in" (min_int_ceo_pg1:2).

There are two points of interest here. First, "the whole team and we [directors] can talk about where we want to be spending our time" and second, the team decided not to focus too heavily on consulting. This highlights that within Minerva (community) all personnel have input into the business strategy (object). This meant that developers and Technical Leads were not just there to perform project-based development work but also to feed into business strategy decisions to some extent (division of labour).

Not only was client interaction and involvement encouraged and nurtured (object); but it was also controlled. It was mostly channelled through the Scrum Master (division of labour), Directors (division of labour), and via certain Scrum practices described earlier. Coaching (object) was practiced by all developers (division of labour) and Directors (division of labour).

Quasi self-managed teams may be an adequate term to describe the observed strategy at Minerva; quasi self-managed because developers often had responsibility for client projects but also worked within higher level constraints, including software development processes and technologies. Developers assigned to a project managed their own inter-project affairs but reconciling interference from management could not be avoided. Directors of Minerva insisted on the sustained ability of developers to think for themselves as opposed to structured, checklist type operations. Management at Minerva ensured that the small workforce have the ability to manage software projects. All personnel demonstrated a propensity towards continuous learning and coaching. This was not seen as a separate job but as an integrated awareness and a characteristic of their everyday project work. There were mechanisms in place that supported coaching and mentoring. Client enlightenment was considered an important strategy and was sustained throughout all projects. However, it was somewhat controlled so as not to interfere with project progress.

6.4.11 Learning practices

Learning practices supported the of transfer of lessons learnt and experiences from completed projects to current ones. Organizational learning was positioned within this knowledge area.

This thesis did not set out to establish the level or amount of learning taking place at Minerva. Instead, practices and instruments in support of learning at the project and at the program management level were explored.

Subj—instr—obj: Scrum techniques such as sprint retrospectives (instrument) allowed for transference of experiences; developer meetings (instrument) allowed for technological knowledge sharing; and to some extent daily stand-up meetings (instrument) complemented learning and knowledge transfer across projects (object).

Learning is traditionally an integral part of knowledge and innovation intensive work like software development. Besides being an integral part of the developer's (subject) daily activity, at Minerva there were practices that were instilled specifically to support learning (object). These included the likes of sprint retrospectives, code reviews, and developer meetings (instruments). The CEO commented on sprint retrospectives:

> "Yes. We do have retrospective meetings at the end of a project but not always. We are a little bit lazy sometimes. But it is valuable to go through. And you've seen some of the metrics that we use, did we go over budget, over time, what was the quality like, was there a whole lot of bugs, was the client actually happy with what was delivered. We look at these things and we look at what we did well, what not so well" (min_int_ceo_pg2:5).

However, this practice was not without challenges:

> "In the last six months we haven't been great at sitting down and identifying strictly on every sprint what went wrong etcetera. Some of the gauges have been quite informal. That said, I think what we have done in situations where we have done brilliantly or done very poorly we have had those meetings [sprint retrospectives]…Based on those meetings we then decided on one or two changes not ten changes differently" (min_int_tl1_pg6:2).

Documentation of lessons learnt on project and technical issues was facilitated through the use of the electronic repository Microsoft OneNote (instrument) and through the social communication medium Yammer (instrument). However, the majority of knowledge appeared to be held tacitly among different personnel (subject). Seeing as Minerva is a SMME, transference of tacit knowledge was constant and at most times was informal (instrument) and facilitated by both a small workforce and an open plan office arrangement.

When questioned about documenting the lessons learned from one project to the next and from one client to the next, the Commercial Director did state that this was done from one sprint to the next and that troubling metrics were acted on if they appeared to be recurring across projects. Learning from client demonstrations (objects) occurred during the sprint review (instrument). Here developers and Directors alike learned about the client business.

Subj—rule—com: Although not enforced as strict policy, at Minerva (community) developer meetings occurred weekly (rule) and all developers (subjects), including the Technical Leads (subject) had to attend (rule).

The following extract from observation field notes detailed an instance.

> I attend a weekly Developer Meeting. All developers attend this meeting. The intention is to update the developer staff on new programming techniques, blogs, sites, methods, functions etc. This is purely a technical session where technical issues and experienced problems are discussed. The meeting is conducted using MS OneNote. MS OneNote is used like a central index which contains scripts, links to forums, links to blogs, classes, code snippets and documentation from research. There are multiple notebooks for different aspects. The focus of this meeting is on new tools, controls, code etc. for the different concurrent projects. This is evidence of continual learning. [Technical Director] leads the discussion and is the chair. (min_obs_pg2:1).

However,

> After meeting, [developer 1] speaks about a problem with decision making regarding this meeting:
>
> "If someone is absent or zones out during the meeting, then they are unaware. There is a problem with carrying through decisions" (Developer 1)
>
> (min_obs_pg2:3),

and

> At the end of the session, [Developer 1] makes note of decisions made in MS OneNote. He tells me this is the first time someone has taken note of decisions made.
>
> "Unfortunately not everyone will be willing to take note." (Developer 1)
>
> After the hour session, I ask [developer 1] about the structure and recording of the sessions.
>
> "One of the problems we have is no scribe. No note-taking. We try to keep these meetings as informal as possible in an attempt not to impose too much structure which would burden everyone." (Developer 1)
>
> I told [Developer 1] that I notice everyone takes notes on their own books, tablets etc. I think that works in this environment due to the highly skilled people who can commit to memory.
>
> "Ja well, attention spans are also a factor. I usually lose focus in the last 15 minutes." (Developer 1)
>
> (min_obs_pg9:2)

Sprint retrospectives were scheduled for each project (rule) and included only personnel (subjects) involved in that project, the Scrum Master, and one or both Directors. However, occurrence of sprint retrospectives was not strictly enforced as detailed by two developers:

> "At Minerva we are good at that. I think sometimes it is not particularly well documented… There is an effort made to learn from previous projects but it is not nearly well documented or a formal process. We do a lot of learning from technical mistakes for which code reviews have been

> extremely useful… For project management point of view I think it's more of a challenge. And it's been noticeable from [Scrum Master] side where it's been repeatedly frustrating" (min_int_dev8_pg33:3).

> "We only started this in the last few months with [Scrum Master] being the project manager. We now have a sprint retrospective. We try to. We not doing it right yet because it's not yet in our pattern. But we are trying to make it a habit of doing it at the end of every sprint. We communicate what went right, what when wrong, what went very wrong to the other teams" (min_int_dev3_pg16:3).

Code reviews (rule) was an important practice for learning in the technical space. Code reviews were held for all developers (subject) irrespective of their project assignments (community). These two developers shed light during the interview:

> "In code reviews you are always learning something new. They teach you how to do something more efficiently or new ways of doing things" (min_int_dev4_pg20:1).

> "Issues are shared, what technologies to use, and what things to look out for. Patterns for programming and methods for speeding up our programming" (min_int_dev5_pg23:1).

Com—div of labour—obj: Every member of the workforce at Minerva (community) was capable of playing multiple roles in support of learning (object).

> "As an individual you never stop learning or constantly learning on that project. But, these are one of the things we try and focus on in our chalk and talk. If there was a new technology implemented or if there was a change in any kind of structure or in the way we did things, part and parcel of the chalk and talk was to expose the rest of the team [to] new advances or new decisions made within the company or new directions that were taken" (min_int_dev6_pg26:2).

A developer or Technical Lead facilitated (division of labour) the developer meeting and introduced new methodologies or technologies that he or she had recently worked with. Similarly, any one at Minerva (community), with the exception of the Scrum Master who is not involved with technical work, could act as a coach or advisor (division of labour) for technical challenges being experienced, thus, advancing the practice of learning across projects (objectives). This could also occur in an informal manner.

The practice of organizational learning took place mainly tacitly with few observed artifacts serving as a repository. Much of the lessons learned from project work and understanding of client businesses is held tacitly. Note taking was not frequently observed. Some developers felt that a short attention span during these meetings was a problem. This information was disseminated as the need arose among the small workforce. Effort was not expended on concise commentary of learned lessons and project experience mostly due to the state of constant evolution of development process.

6.5 Summary

This chapter described the data analysis. AT's conceptual components and its epistemology of mediating actors formed the lens for exploring the interfaces between program management practices, SMME characteristics, and Scrum practices. Interfaces were viewed as a series of relationships, each constituting an activity system endeavouring towards successful outcomes within a program management knowledge area.

The following program management knowledge areas were identified via the literature review and reported on separately for both Athena and Minerva:

a. Staff allocation
b. Coordinating activities
c. Cost estimation
d. Human resource management
e. Infrastructure design
f. Program monitoring and reporting
g. Planning
h. Client liaison
i. Schedule risk identification and mitigation
j. Alignment of practices with business strategies
k. Learning practices

That said, some new areas and constituent practices emerged in the fieldwork. These included practices such as preaching and educating the client in Scrum practices, being selective in taking on clients, self-organizing teams, and developer accountability for projects.

This chapter laboured towards unearthing insights for the research questions why do the characteristics of software SMMEs influence program management practices?, and how do the key activities of Scrum influence program management practices? The major contribution of this chapter is an interpretation of the influences of SMME characteristics and Scrum on program management with the resultant program management knowledge areas. This interpretation was crafted using the theoretical concepts of AT. The next chapter focuses on more in-depth interpretation of the data with the objective of establishing common practices within the program management knowledge areas and unearthing the nature of program management.

CHAPTER 7: INTERPRETATION OF THE DATA ANALYSIS

7.1 Introduction

This chapter interrogates the data analysis presented in the previous chapter and provides a view of the emergent and heuristic (or rule of thumb) nature of the relationship between program management practices, SMME characteristics and Scrum practices. Through a process of thematic analysis, program management practices common to Athena and Minerva are described within each of the contextual program management knowledge areas. A further refinement to the data analysis was afforded via concept analysis. Concept analysis is introduced and described in this chapter and allowed for a representation of a temporal dimension within the findings. The eventual findings and major contributions of this study are presented in tabular format for each program management knowledge area. Each table is representative of a theoretical construct portraying the interface between SMME characteristics, Scrum and program management practices.

The chapter begins with a review of levels of theorizing first introduced in Chapter Three. It then introduces and defines concept analysis and the importance of its usage to refine data analysis. The next set of subsections present the theoretical constructs for each knowledge area, namely staff allocation, coordinating activities, cost estimation, human resource management, infrastructure design, program monitoring and reporting, planning, client liaison, schedule risk identification and mitigation, alignment of practices with business strategies, and learning practices. The chapter then presents a view on quality management of program management practices, before ending with a brief discussion of the stability of the observed program management practices derived from the concept of the zone of proximal development.

7.2 Review of the levels and categories of theorizing utilized in this study

Chapter Three introduced the proposed levels of theorizing by Llewelyn (2003) and indicated categories of theory suggested by Gregor (2006). Recall that this study begins with Llewelyn's Level 1 theory based on the literature review, progresses to Level 3 when program management practices are better understood in the Scrum context, and finally makes a contribution at Level 4 where work practice is not merely seen as a technical endeavour but as practice within the given social and/or organizational setting. This study as a whole best

satisfies Gregor's category of explanation theory. In order to satisfy Level 3 requirements, common emergent practices from both case studies need to be identified and summarized. This is sometimes referred to as cross-case analysis (Yin, 2012) and this is the intended purpose of this section.

Eleven program management knowledge areas formed the containers for practices at both case study sites. Fieldwork revealed similarities and revelations for each. As discussed in Chapter Two, the available current literature provided little advice in the way of understanding how program management should be practiced in the context of Scrum software development. In light of this deficiency, the data analysis sought to uncover the interaction between the SMMEs, Scrum and program practice. However, a temporal disposition of the interpretation of findings is required. For this purpose support was found in research in the arena of health sciences.

7.3 Using concept analysis to model the interpretation of data

> "[…] concept analysis aims to uncover the attributes of a concept by calling attention to its common use in current literature, providing a foundation for further inquiry and promoting continual concept development" (Russell, 2012).

Concept analysis adopts a stance whereby theories should be developed via a hermeneutic process of clarifying concepts (Welford et al., 2010). In the research field of health sciences, particularly in nursing, rigorous work has been done in the area of concept analysis as a vehicle for theory development. Good commentary on its evolution can be found in Welford et al. (2010) and Russell (2012). These and other authors position Wilson (1963) as the architect of modern concept analysis, with significant revisions by Walker and Avant (1988) and then by Rodgers (1989). Concept analysis is popular in desktop research in that it provides a mechanism to synthesize concepts from findings of multiple published studies. For more general examples see Tjale and Bruce (2007), Zulkosky (2009) and Nuopponen (2010); and more specifically in information systems research, see Burton-Jones and Gallivan (2007) and Burton-Jones et al. (2011).

A standpoint is made in this study that certain stages of this process of concept analysis can be applied to fieldwork data as well. Although rigorous AT analysis was performed in the previous chapter, the additional refinement afforded by concept analysis is of interest; particularly the way epistemology of antecedents, attributes, and consequences are identified from contextual literature. Table 13 defines these epistemological terms:

Table 13: A description of concept analysis epistemological terms, synthesized by the researcher from the literature.

Term	Description
Concept	Terms used to describe rather than provide definitive definition; for example, a dictionary shows how words are used in sentences from which the reader may conjure a meaning.
Antecedents	Activities or conditions that must arise prior to concept occurrence. More simply, what happened before the concept?
Defining Attributes	Unique characteristics that differentiate one concept from another related one.
Consequence	Events that arise as a result of the concept, which often drive new ideas or research; that is, what happened after the concept?

Once these components have been identified, Rodgers (2000) suggests presentation of the concept using a framework called a model case which showcases antecedents, defining attributes, and consequences. A model case can be described as:

> "[…] the best understanding of the concept at the time of analysis. It should include all of the defining attributes and present an undisputable and unequivocal illustration of the concept in the context of which it is being examined" (Welford et al., 2010).

Johns (1996) furthers the original implementation of concept analysis by adding the dimensions of process and outcomes. Johns argues that concepts also exhibit characteristics of a process and can be either static or dynamic. She argues that the initial models by Rodgers (1989) and by Walker and Avant (1988) fall short by firstly, not identifying concepts as processes; and secondly, by not viewing concepts as outcomes at some point in time. This thesis sees the usefulness of the concept analysis model particularly the epistemology of antecedents, defining attributes, and consequence and the temporal impressions of a concept proposed by Johns (1996). Table 14 illustrates this tailored model used as a framework for the interpretation of data analysis in this thesis.

Table 14: The framework used for data analysis. Derived from concept analysis model of Johns (1996)

Antecedents		Defining Attributes	Consequence
SMME characteristics	Scrum practices	Key common practices for program management knowledge area	Emergent nature of relationship

This thesis endeavours to provide a representation of the interfaces between SMME characteristics and Scrum practices and how they impact on program management practices

within a Scrum software development environment. The table above highlights that the observed SMME characteristics and adopted Scrum practices at the time of fieldwork constitute what served as antecedents for the defining attributes, which are program management practices. Defining attributes are the unique practices that separate each of the knowledge areas investigated in this study. Consequences are the emergent nature of the relationships between SMME characteristics, Scrum practices, and program management practices.

7.4 Motivating the choice of concept analysis

A concept undergoes continuous development shaped by time and context and is dynamic in nature (Russell, 2012), and the model is emergent rather than predefined (Welford et al., 2010). This notion meshes with the descriptive nature of this thesis, since this thesis has no predetermined preconceptions or biases towards what might be observed. Instead it embraces the emergent nature of a descriptive study. Although the term system is not an adequate synonym for activity system, the principle behind this quote is applicable: "a system is not something presented to the observer, it is something to be recognized by him" (Skyttner, 1996).

Another key reason behind the selection of concept analysis is the temporal dimension that Johns (1996) adds to the existing concept analysis models. Her perception of concept and outcome provides a temporal frame of reference for the program management practices explored in this thesis. For Johns (1996), a concept evolves over time, which she depicts as a stepwise process with feedback loops. Also, a concept can be captured at some point in time which she describes as a *snapshot*. This notion of a snapshot in this thesis has a slightly wider angle in that it refers to common practices of program management across two case sites within some point of time in an era of agile software development in these SMMEs.

The subsections that follow represent a snapshot of each knowledge area at the time that the fieldwork was done. This is deemed useful since AT analysis revealed that the majority of practices in each knowledge area pass through a zone of proximal development. Each subsection concludes with a table outlining the antecedents, attributes and consequences of that knowledge area.

7.5 Staff allocation

There were similarities and a few subtle differences to staff-project allocation at both case study sites. Athena practiced cross-project staff allocation; meaning that a developer was assigned to one or two primary projects; and fixed allocation, whereby a core group of developers stayed with the project until the end. Developers either volunteered or were asked by management to *help out* projects in danger for a short duration before returning to their assigned projects. Minerva employed a two person core project team with additional developer reinforcements only when the need arose. Both SMMEs seemed comfortable with their approaches; although both also admitted distance from perfection and the need for improvement. Furthermore, the nature of a small workforce within the SMMEs often led to ad hoc staff allocations to curtail short-term challenges within different projects. The observed nature was one of fluidity with the absence of conformance to strict structure and rigidity towards staff allocation.

Maintaining awareness of developer assignments appeared as an uncomplicated endeavour. Although software tools did provide support, the small workforce and close interaction permitted easy tracking of developer-project allocations.

Developers were mainly allocated to projects by management or senior personnel. Preferred selection criteria emerged as being availability of the developer, prior experience, and engagement with the client so as to try to preserve client relations and improve on knowledge transfer, and technological expertise. Experienced developers or developers with greater technical skill doubled as mentor or coach for the rest of the team. In the case of additional resources, the developer's familiarity of the system and/or the client business was considered. There appeared to be a period of time before the unfamiliar developer became acquainted well enough to start contributing on the project. As a means of alleviating this challenge, these developers were usually asked to code more stand-alone software components in the beginning and then graduated to more complex and integrated parts of the product. Technical unfamiliarity did not present a challenge due to just a handful of architectures that underpinned most developed software systems used. The compiler technologies used to develop the software were mostly common across projects.

There were instances in both cases where developers requested to be assigned to particular projects due to their interest or eagerness to increase competency in that domain area or technology. The opposite was also noted in a few instances whereby a request was made for

removal from a project. These requests were often granted due to the lightweight management structure and employee propensity to learning. Both served as mechanisms for nurturing a motivated work force within the SMME.

Table 15: Staff Allocation antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Staff Allocation Practices**	**Emergent Nature of Relationship**
Limited resources for a having separate client relations management roles.	Close client collaboration was a closely adhered to underlying agile principle.	Maintained client relations by assigning developers familiar to the client and/or client business. Developers were selected for a project based on parameters that included prior interaction with client and skill set.	Allocations exhibited an undercurrent of preservation of client relationships.
Lightweight management mechanisms supported shifting allocations. Small workforce consisting mostly of highly skilled software generalists.	Allocation awareness was facilitated via daily stand-up meetings.	Developers had in some cases leeway to reject or request allocations. Additional developers were assigned to projects as those project's progress dropped.	There appeared to be a contradiction between developer choosing their own projects and management allocations. Allocations were mostly done by management and they appeared to have good awareness of skills and competencies of their workforce.
Small workforce fostered easier awareness of allocations.	Allocations were made visible through electronic and visual artifacts and daily stand-up meetings.	Maintained an awareness of staff allocations.	Small workforce, electronic support tools and visual aids were instrumental in order to maintain awareness of project-staff allocations.

7.6 Coordinating activities

Coordination practices included task planning, task allocation and progress monitoring, and were done mainly by the developers at the project level. This mannerism was aligned with the agile principle of self-coordinating teams. It also satisfies the SMME characteristic of a more lightweight management process. Daily stand-up meetings, with their constant broadcast of project progress status, served as the primary instrument in support of this practice. Athena employed Kanban boards to support individual project daily stand-up meetings, while Minerva used only a spreadsheet reporting on budget and had a single daily stand-up meeting for all current projects. This proved that visual aids were not a necessity for project awareness within a SMME setting, but rather a preference.

Scrum artifacts in support of this practice included the product backlog and release plans. Developers selected their own user stories for coding from the release plans. There were no observed strict structures or guidelines for this set of practices, nor was there constant interjection from management. Instead, developers naturally sought a balance between venturing into new areas of software development and technologies, and selecting tasks within their comfort zone so as to maintain acceptable project velocity. This is proof of a slightly higher inclination towards a generalist nature instead of a specialist one for the developers from both sites. Management did intervene whenever a project lagged behind schedule. Intervention permutated in the form of emergency type workshops and meetings aimed at analyzing operations, organizational learning, and bringing the project back on track.

In both cases self-coordination of tasks instilled a sense of ownership and responsibility within developers for their projects. Management was adamant about self-organizing even with the potential risk of slowdown in project progression. This slowdown is only natural given the presence of learning curve and adjustment of developer to this peripheral responsibility.

Table 16: Coordinating Activities antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Coordinating Practices**	**Emergent Nature of Relationship**
Lightweight management permitted developer self-coordination of	Visual aids and software support tools extensively supported planning	Planning of daily project tasks, allocation of tasks and progress monitoring was done at	Multi-level decision-making. Driven by real-time

project tasks. Flat organizational structure facilitated the communication of daily project tasks.	and task allocation. The product backlog and release plan were in place for task coordination. Open plan office supported informal communication.	project level by developers .	accurate reporting of projects. High dependence on continuous update of project status via software support tools and visual aids.
Small workforce with varying levels of expertise and experience.	Pairing of senior or experienced developer with a novice or a new developer helped with coaching.	Management intervened in coordination of tasks when project lagged behind schedule. Mostly s*oft* intervention in the form of reflective practices and of coaching and mentoring.	Although self-organized, management did intervene to harmonize program management level. Management sought to help development teams to help themselves. *Soft* intervention takes its lead from Lean guideline of continuous reflection in action and redesign of process.
A small number of human resources within a close knit environment.	Scrum guideline of small development teams per project.	Developers had a strong sense of responsibility and ownership over their assigned projects.	There wasn't much need for strict control as developers appeared self-motivated.

7.7 Cost estimation

The overall budget and delivery date for a project was negotiated by management and the client at the onset of the project. However, it was evidenced that these financial constraints rarely remained the same for the duration of the project. In both cases the SMMEs aimed to deliver a business solution, not just a software system. Coupled with limited flexibility in software requirements, an additional layer of complexity was placed on the estimation process due to the absence of concise software functionality requirements early in the project. To some extent Scrum was able to absorb this complexity via short delivery cycles and multiple sprint reviews which engaged the client frequently in the development process. Changes were negotiated in the form of later delivery dates or higher cost for additional

functionality requirements. Both SMMEs were very selective of clients and only chose work where the client believed in the principles and the effectiveness of Scrum. Most clients acknowledged the required levels of requirements negotiations and accepted this as a natural occurrence within Scrum projects.

In keeping with the characteristics of a flat hierarchical structure, management and senior personnel of both SMMEs actively participated in project level work; that is, in software development work. This not only helped increase productivity but equipped them with better understanding of the ability and of the skill of the workforce. This immersion seemed to help management's tacit knowledge of the SMME's workforce ability and skill, which was necessary and paramount for deriving budget and delivery dates for projects. Software support tools provided velocity figures for projects, a supplementary indication of developer's ability. The exception was the CEO of Athena who concentrated purely on enterprise management practices.

The primary metric for cost estimation in Scrum was the user story point. The user story point was a perception of the complexity of a functional requirement and, by extension, an indication of the duration of development. Developers assigned to a project were in charge of coding the user stories within budget and within the delivery date. This denoted a certain level of transparency of the financial aspects of the projects. Instances of mismatches between the delivery dates set by management and the team's ability to meet it, resulted in a margin of inaccuracy within estimates. This stemmed mostly from the developers wanting to be on the *safe side* in their estimates. Budget and delivery dates were renegotiated in some instances where the client was willing. Most of the time teams were urged to work faster. However, this *working faster* didn't always translate to simple exertion of more effort. It often triggered reflective and change practices. This is discussed in more detail later in section 6.12.

The most interesting revelation in this knowledge area was the continual adjustment to and subsequent reporting of budget usage and schedule of remaining work. Unlike traditional software engineering where the budget is generally detailed and cast in stone at the onset of the project, in these cases budget was reported daily. This reporting often led to adjustments in work plans. Stated differently, the budget directly influenced the work plans for a project on a daily basis. Clients were made aware of the Scrum process; hence, most understood the logic behind it and accepted the need to adjust budget and in some instances the delivery

dates. The SMME's lightweight internal processes coupled with non-rigid structure provided suitable dynamism to absorb such fluidity in everyday work practices.

It was quite clear in both case sites that cost estimation was in a state of development. This is aligned with the current state of the field of Scrum and by extension, agile software development at the time of writing.

Table 17: Cost Estimation antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Cost Estimation**	**Emergent Nature of Relationship**
The small workforce afforded management intimate understanding of developer competency.	Product backlog indicated prioritized work items. Short delivery cycles via sprints meant cost estimations could be re-evaluated often. Daily stand-up meetings and burndown charts facilitated in reporting.	Initial delivery date and cost for each project was negotiated by management and client. Cost was adjusted according to project functional requirements changes. Cost estimations were continuously adjusted to accurately reflect financial wellbeing across projects.	Represents a departure from highly detailed upfront estimation. Instead negotiation was seen on several occasions and since both SMMEs were selective of clients, clients seemed to accept this nature. Cost estimation is an area in a high state of evolution.
Developers performed peripheral work of cost estimation for their assigned projects.	Sprint release planning, visual aids, and software support tools provided vital information during cost estimation.	Cost estimation of user stories was done by developers.	Cost estimation was done by developers at the project level but the custodians of profit margins were at the program management level. Loss in one project due to new client, developers, or technology, was tolerated against the backdrop of overall profitability from other projects.
Limited tolerance for financial loss	Daily allocation of work tasks via daily	Real-time reporting of budget.	Budget affected work plans on a daily basis

drove continuous (daily) optimization of work plans.	stand-up meetings.	Work plans were adjusted on a daily basis to compensate for changes in budget.	which relied upon the ability of real-time project reporting.

7.8 Human resources management

In the case study SMME development environments studied, software developers were the commodity that most often led to conflict. The practices in this knowledge area involved the initial allocation of developers to projects, and then focused attention on assigning developers as short-term reinforcements to needy projects. Adding more developers to projects was an ongoing, *juggling act*. A persistent challenge with this practice was in the abstract art of avoiding too much disruption to concurrent projects, while preserving the team's established working relationship. Good team relationships typically ensued, allied by mutual understanding of each team member's strengths and weaknesses, which in turn facilitated smoother coordinating and planning activities. The small workforce of the SMME fuelled this understanding. Management scheduled events for improved working relationships among the workforce; such as outside workshops, informal get-togethers and team building exercises organized by outside vendors specializing in these areas. Within the close knit SMME environment, poor working relationships were deemed counterintuitive for the organization.

Practices sought to prevent projects from reaching too high a velocity. With too high a velocity, it became difficult for a project to adjust to changes. This is a hallmark of Scrum and other agile process models. Too much manpower is generally considered a wasteful expenditure of resources due to a perception of software development being a creative and cognitive process. Problem solving and researching techniques and alternative methodologies to software design and development were practices that required time. Human resources had to be diverted away from projects entering this state until the team was confident enough to increase the development speed.

More often than not, developers did not harbour a notion of segregation along the lines of *that's not my job*. When developers saw a project falling behind and they felt that they could assist through their technical expertise or by adding more muscle, they did so voluntarily and in some cases even after hours. This camaraderie and sense of esprit de corps was mostly affiliated with a close knit relationship between employees and a culture of feeling responsible for the overall well-being of the SMME.

Table 18: Human resources management antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Human Resources Management**	**Emergent Nature of Relationship**
Small workforce enabled intimate understanding of developers' competency and their current assignments. Limited developers resulted in a constant scuffle for optimized usage.	Daily stand-up meetings, team velocity figures, and burndown charts served as indicators for project progress. Informal communications relayed data on progress or bottlenecks.	Developers were shuffled between needy projects and their formally assigned projects. Reassignment of developers often resulted in mitigation of regression effects in other projects. Developers were diverted from their projects that entered wait states.	The assignment of developers to needy projects did not follow a predefined structure and appeared to be based more on reflexivity. An almost ad hoc nature of practice was observed. At level of program management there was an awareness of the developers' current assignments, coupled with knowledge of their expertise, allowed for commissioning of the *right* mix of skill and experience for the different projects.
Lightweight processes both catered for and absorbed shifting work assignments. Developers were expected to work in more than one domain or technological area.	Agile principle of embracing change meant that a lot of effort could potentially be wasted if velocity was too high. Software support tools closely supported measurement of both developer and project performance.	Avoided reaching too high a velocity within any project.	The pace of work was maintained within the proverbial *sweet spot;* but this was sometimes difficult to sustain.
Limited tolerance for poor inter-team working relationships.		Preserved established relationship within project teams, but sometimes jeopardized by cross project movement of developers. Team building was	A close knit community which was helpful towards each other and who typically did not want to see others fail for their own personal

		organized and conducted by outside companies. Established training needs, training requests, and organized training for employees	gain. Strong sense of esprit de corps.

7.9 Infrastructure design

Infrastructure design was the only activity system where practices demonstrated a state of stability. This practice focused on providing an environment conducive to agile software development. This included establishing sustained close interaction between teams and between client and teams, making project task status and accounting transparent, and ensuring adequate project and work support tools. In both cases conventional Scrum practices prevailed. The only major difference was the denouncement of physical visual aids at Minerva.

Table 19: Infrastructure Design antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Infrastructure Design**	**Emergent Nature of Relationship**
Small workforce afforded intimate work relationships. Open plan office germinated informal communication and tacit knowledge transfer.	Required informal communication for knowledge transfer in the absence of extensive documentation. Visual and electronic aids disseminated project information.	Ensured an environment conducive to Scrum software development.	Practice-driven by developers and managers alike. Appeared very stable with no change during the course of the fieldwork.

7.10 Progress monitoring and reporting

In both cases the Scrum Master was the focal actor for progress monitoring and reporting practices. He collated daily progress reports using a plethora of practices; such as daily stand-up meetings, software support tools and informal communication. It was evident that a strong sense of trust presided over the need for strict Pacioli double entry type reporting. Once reconciled a holistic image of all project progress was made available to management and developers alike. This information was easily digested and fed among the small workforce.

Agile, and by extension, Scrum-based metrics of user story points and unit test percentage featured extensively as barometers for monitoring progress.

Progress monitoring and reporting was done in real-time. It was possible to provide on-demand, and up-to-the-minute, reports on project progress. However, there was heavy reliance on the peripheral task of developers timeously updating task status on the software support tools. Although this was not an overbearing challenge, on a few occasions the Scrum Master had to remind and urge developers to complete this task. Close and constant informal communication among the developers and a flat organogram supported the Scrum Master with this cause. This real-time reporting served as radar and allowed for project teams to adjust their work patterns. It also triggered interventions from management at early sightings of lurking danger.

Scrum project artifacts, such as burndown charts produced by the software support tools and budget charts prepared by the Scrum Master, were easily interpreted by personnel and provided a clear indication of a project's health. The Scrum Master prepared budget burndown representations using excel spreadsheets. Athena relied on visual aids such as the Kanban board for project feedback, but these were not employed at Minerva. This production and presentation of charts was formalized through the daily stand-up meetings but could be viewed at any point. Apart from the monitoring of economic metrics like progress and schedule, these were not the only attributes that served as measures for the wellness of the project. Other noted attributes included client satisfaction as well as developer well-being and state of mind while progressing through and after a project. Management and senior personnel situated these soft metrics high on their agenda. All personnel were encouraged to voice their opinions and engaged management in the search for establishing a mutually beneficial working relationship. With the exception of the developer who joined and resigned from Athena within a short period, all other developers from both case sites clearly stated that they enjoyed their work. They further felt that they had a direct influence on changing operations at the project level for improved efficiency.

Both SMMEs were continuously striving for improvement in this knowledge area with processes that have changed relatively quickly over the last two years. In both cases, progress monitoring is in a state of evolution with neither camp satisfied with the accuracy of prediction. Reporting to both the team and client was deemed a seamless task.

Table 20: Progress monitoring and reporting antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Progress Monitoring and Reporting**	**Emergent Nature of Relationship**
Responsiveness and dynamism of internal practices required support from similarly flexible monitoring and reporting mechanisms.	Burndown chart, release plan, product backlog, software support tools, and visual aids provided real-time progress reporting.	Monitoring and reporting of progress of different projects was done in real-time and on-demand. Developers had to update task status timeously to enable accurate reporting.	Progress monitoring and reporting was in real-time but the Scrum master had to *remind* developers of peripheral task of updating progress. Both cases showed that this practice still was in a state of evolution.
A flat organizational structure facilitated transfer of knowledge. Large amount of informal communication.	Formal reporting took place mainly during the daily stand-up meetings and during sprint reviews. Visual and electronic aids were in place for informal reporting. Open plan office supported informal communication.	Formal progress reporting for client and development team. Informal reporting was not practiced per se, rather it existed as an inherent part of day-to-day work.	There was no strict monitoring and reporting; instead, much of this practice was based on trust i.e. accurate feedback from developer at the project level. Lean principle of making work visible was adhered to.
There are limited resources for marketing and advertising, hence reliance on *word of mouth* and recurring clients. There was a need to sustain the workforce due to potential of irrecoverable loss in lost knowledge and experience.	Practices such as sprint retrospectives provided a forum for open and uncriticised dialogue to highlight experienced problems.	Monitored *soft metrics such as* client enlightenment and developer well-being during project.	Practice shows close alignment with the strategies proposed by Denning (2010b) and Martin (2011) where it was established that project management entails more than economic measurements.

7.11 Planning

The Product Owner played an initial role in planning practices by translating his vision of a software system into a prioritized list of requirements. The underlying premise was that the

client has more intimate knowledge of his business and the business processes that the software is trying to emulate and coexist with. The Product Owner was helped by assigned developers and management in completing this task. In both cases however, clients were informed from the beginning that the project's aim was to provide business value and not simply supply a software product. This approach represents a breakaway from classical software engineering approaches in that the development team attempts to unravel the intricacies of a client's business during the early stages of the project. Tension arose when the Product Owner tasked with creating the product backlog did not have intimate knowledge of the business, and thus did not create concise and completely necessary user stories. However, the product backlog was not cast in stone at the onset of the project and the Product Owner could adjust or *groom* the priority as requirements crystallized. The dynamic nature of SMME internal processes allowed for this level of tailoring. In some instances user stories were removed or added. This usually occurred after a sprint review session or after changes related to the client business. Development teams considered the consequent activity of adjusting project level plans as a natural occurrence. There were no instances of any SMME personnel attempting to sway the Product Owner in light of less or lowered complexity in work. Instead, establishing a close working relationship was the key objective.

Planning at the project level was done by the developers via Scrum practices such as release planning, sprint planning and allocation of tasks on a daily basis. Management did not actively participate in planning but did indirectly influence the planning in their passive roles as coaches. The observed principle was that a developer understood their capability and therefore he or she was best located for planning project work and for staying within reasonable project velocity. The informal management mechanisms of the SMME fostered this practice. These practices were also aligned with the Scrum principle for lightweight planning.

Interviews showed that in some instances developers *padded* their estimates to compensate for unknowns such as a new business domain or new prototype technologies. This planning had to be done before actual coding started. The Scrum Master utilized probability settings in the software support tools to create a *cushion* for budget and delivery schedules despite *padded* estimates by developers in some instances. Also, projects that were comprised of team members that hadn't worked together before, or coding with state-of-the-art technology presented challenges for accurate estimation. Playing poker was a means of mitigation in some cases whereby a team of developers worked on establishing an estimate.

Once planning was done, reporting of these plans was done via software support tools. Release plans and product backlogs were constantly updated and made visible. Project management support tools facilitated in this process; and at Athena but not Minerva, visual aids were employed for additional reporting. Overall, awareness of these plans was easily maintained given the intimate work force of the SMME; so much so that developers from other projects easily detected problems and volunteered their assistance, sometimes in their free-time without any announcement or appetite for recognition.

In line with the mantra of agile, planning was not too forward looking nor was it highly detailed. Some clients did not always subscribe to this rather outward haphazard approach but both SMMEs were very selective of their clients. Clients that understood and bought into the process of developing software via Scrum were more open to renegotiating functionality and/or delivery schedule.

User story estimates were the major unit of measurement for planning. User story estimate was a perception of complexity and in turn led to an indication of what amount of time was needed for development. Angst surfaced when developers had difficulty in interpreting, or when they misinterpreted user stories, which jeopardized their plans to some degree. As a means of mitigation, more than one developer worked on user story estimates such that deliberation ensued. Interviews revealed some calls for developers to have a more active role in the analysis of client business and the development of the product backlog.

The practices of planning in both cases appeared to be in a state of evolution and far from an exact science. Nonetheless, in both cases the ability to accurately plan was deemed an important quality characteristic as well as vital for maintaining good client relations.

Table 21: Planning antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Planning**	**Emergent Nature of Relationship**
Close engagement with client to establish good working relationships. Most developers considered accurate planning as a direct attribute required for good client relationships.	Adhered to Scrum principle of close client interaction.	Inducting and coaching client or Product Owner and helping them to prioritize business needs. Assisted the Product Owner with crafting the product backlog.	Client is involved as an active change agent within the Scrum project and not simply a recipient of the software product.
Lighter control mechanisms required developers to set their	Product backlog planning and grooming afforded the client with	Groomed the product backlog to align it with most recent understanding	Planning is done at the project level by the developers but was fine-

own milestones but within project margins.	a means to capture their product vision. *Playing poker* with multiple developers sought increased accuracy of estimation and resultant planning.	of requirements. User story estimates were done by developers and served as a major unit of measurement. Sprint planning, release planning of work for the next 3-4 weeks. Daily task planning.	tuned by the program management level on a day-to-day basis to better suit the program management space. Planning was driven by the user story estimate. It was entirely possible that a different team could have arrived at a different estimate and subsequent plan. Plans never seemed cast in stone and were sometimes immediately adjusted to compensate for different understanding of, and additional, requirements. Fixed budget set by management served as a stabilizing agent to otherwise tumultuous plans. Developer padding and probability settings indicated signs of unsettled and yet to be improved upon planning activities.
Small workforce with close interaction helped to communicate plans .	Software support tools and visual aids communicated plans via product backlog and release plan.	Reporting of plans to project teams and management.	Practice of communication of plans seemed effective and relatively effortless.

7.12 Client liaison

Clients were not perceived solely as external stakeholders in the process of software development. Instead, clients were actively engaged as contributors as well as change agents. On the surface this practice satisfied the Scrum principle of close client interaction but also infiltrated a little deeper. A major observed point of interest was the practice of preaching, educating and inducting the client into the Scrum process. By getting the client to better understand software development through the lens of Scrum, trust appeared to be more easily harboured. This was perceived as an important first step towards good client relations as many developers felt that most potential clients were still heavily biased towards the well-

entrenched waterfall or classical software engineering management paradigm. However, this was an undertaking that sometimes required the exertion of copious amounts of effort from the SMME's side.

Another formal client engagement practice was the sprint review. This was done at the end of the sprint and afforded an opportunity for the client to criticize and comment on the evolving software system. This Scrum practice served as a fit-for-use type of test for the development team and for management. At the project level the development team could adjust the plan and economies of scale based on this client feedback. These practices were enabled by SMME characteristics of high dynamism and responsiveness to changes in the environment, in collaboration with Scrum's short delivery cycles and its propensity to deal with undulating project level objectives.

Reporting of project progress to clients was primarily undertaken by the Scrum Master. For the purposes of queries on functional aspects of the software, a controlled communication procedure persisted between development team and client, with the Scrum Master acting as the mediator. This existed in the Scrum Master's portfolio primarily due to the regressive cost and schedule implications in the program management space arising from changes to functionality. There were software tools in place to facilitate client-developer communication channels but these were not consistently utilized.

Although a priori Scrum principles recommend an onsite client representative with the mandate of functional requirements clarification, only one project was observed with this in operation. In the majority of observed projects clients simply could not spare an employee to serve in this role. Instead, a plethora of contemporary ICTs and social media services served to bridge this communication chasm. There were instances of delays due to waiting for clarification but these delays were absorbed without serious consequences to the schedule.

Key observed guiding philosophies for practices within both SMMEs were the notions of client enlightenment and maintaining client trust. The ethos of the client *enjoying* the interaction with the SMME flowed through each practice in this knowledge area. The SMME strived to establish and nurture a long-term relationship with recurring work from the same clients. With the lack of a stand-alone marketing arm, this was an important notion for the sustainability of the SMME.

Table 22: Client liaison antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Client Liaison**	**Emergent Nature of Relationship**
Maintained close relationships by striving to achieve higher levels of trust with clients.	Adhered to the Scrum principle of close client interaction. The Product Owner was an active participant in the project.	Advocated for Scrum processes and educated the client in its basic operation.	Client was positioned as a change agent. Client was educated in Scrum despite the additional overhead of this activity. This indicates the value of client buy in for both product and process.
Responsiveness and dynamic nature of practices provided for frequent client feedback and quicker realization of resultant change to software requirements.	Sprint review was the major formal mechanism for eliciting client feedback.	Created a periodic opportunity for the client to view, suggest improvements and criticize the software.	Client was given the opportunity to give feedback on projects periodically instead of only at the end. Periodic feedback from individual projects allowed for adjustments to be made to program management plans in an attempt to avoid escalated disruption.
Informal communication mechanisms were somewhat stifled in extension to the client.	Scrum Master played the role of communication intermediary between development team and client. Onsite client representative not always available for projects.	Structured and encouraged communication between client and project team. Technical queries were clarified via onsite client representative (only one observed instance).	Informal communication flourished within the project teams. However, controlled communication persisted between client and development team.

7.13 Project schedule risk identification and mitigation

The fieldwork focused on risk and risk mitigation of project schedule and the resultant risk to budget. In these SMME development environments, schedule and budget risks were never completely averted and all projects were observed being impacted by some mitigation process. Causes of risk came from numerous quarters; such as unclear requirements from

clients, incorporation of new and complex technologies, an unfamiliar business terrain, or even inaccurate estimates of required development work.

Primary Scrum artifacts for risk identification were the burndown charts and budget spreadsheets. The Kanban board at Athena supplemented identification of slow moving progress. Daily stand-up meetings served as an important conduit for progress monitoring. Project management support tools also allied in this practice as did informal communication. The SMME characteristics of close interaction and flat hierarchical structure often meant that management had their *ears on the ground*. Scrum practices, like sprint retrospectives and other formal meetings, promoted practices in search of solutions in mitigation of risks at the project level.

By now it is well established that a primary metric in Scrum-based projects was the user story point. Inaccuracy in user story point projection directly contributed to risk in both schedule and budget inaccuracies. Given the SMME characteristic of small tolerances for budget overrun, risk mitigation was a revered practice. The most direct form of mitigation was having more than one developer assign points to a user story which forced deliberation before settling on common ground. However, this practice was generally of a dynamic nature. As the team got better at developing software with new technology, or grew more familiar with a new business domain, the estimates decreased. Hence, teams got better at completing tasks and should have taken less time to get to completion, resulting in a margin of safety towards the end of the sprint. This was not observed outright mainly due to short duration of fieldwork. Mitigation also included applying probability settings for estimates, an adjustment that was applied by the Scrum Master to the estimated user story points. This adjustment served as a reserve or safety margin. Despite this practice, developers deliberately inflated estimates to compensate for unknowns. Pre-project meetings dedicated to deliberations over risk were also conducted.

Schedule-based risk caused by misunderstood or late client functionality requests often triggered negotiation over more budget or later delivery. Due to clients buying into and trusting the Scrum process, this risk was mostly mitigated by the client agreeing to the later delivery date or to a higher cost. That said, there were a few instances where the developers misinterpreted functional requirements and the client was not willing to renegotiate. The result was a slight loss in profit for that project. At the program management level this loss was leveraged by other more financially healthy projects.

Despite its constant presence, risk was never allowed to escalate to an out of control level. One of the senior developers made reference to a recent South African Government software project that went over budget by R25 million and overschedule by 3 years. He opined that early warning from constant monitoring of project progress could help avoid such tragic outcomes (min_dev2_obs_pg13:3).

Developer teams were responsible for mitigation of schedule risk. Management made it clear that developers were responsible for their own projects, including resolving adverse effects that were invariably caused by schedule-based risks. On the other side of the field, there was an instance where the team itself was the cause of risk. The inability for a team to work together harmoniously was unearthed as the root cause of delays. Finally there were risks that were completely out of range. An observed example was when the client side suffered a major system failure while the developers were getting ready to interface newly developed software with existing systems.

Table 23: Project schedule risk and mitigation antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Project Schedule Risk and Mitigation**	**Emergent Nature of Relationship**
Small workforce and non-hierarchical structure facilitated in dissemination of project status information.	Visual and electronic aids projected deviation and potential deviation from schedule through burndown charts, Kanban boards, and budget burndown charts. Daily stand-up meetings communicated schedule and budget usage. Risk of schedule overrun was always present due to the Scrum principle of embracing change coupled with juggling of developers.	Projects at risk of schedule overrun identified at an early stage to avoid escalation. When the possibility of schedule overruns arose due to changed functionality, cost and delivery dates were renegotiated with client.	Risk was never seen escalating to an out of control or to a *dangerous* level. This was mostly due to the lightweight nature of processes and early warning mechanisms.
Both SMMEs had a low	Pair programming and	Strived for more	Learning occurred

tolerance for overspending.	mentoring was practiced in an effort to increase accuracy in estimation. Software support tools had built in features to allow for probability adjustments of schedule forecasts.	accuracy in user story estimation. Probability settings were applied to derive optimistic and pessimistic schedule project forecasts.	through Scrum practices and as a result estimation improved as teams grew more familiar with their projects.
Lightweight management processes meant that project level risk and mitigation was shifted to the developers. Lightweight internal processes could be adjusted in accordance to crisis alleviation.	Sprint retrospectives and special *crisis* intervention type workshops were held to prevent escalation of risk.	Management provided the space, opportunity and a margin of tolerance for developers to build their capability to resolve risk within their projects. Carried out interventions and reflective practices to better understand risk, possible causes of that risk and to craft resolution plans.	Risk was never viewed as infinite due to a developer's increasing ability to predict and prevent its occurrence. Program management aimed to communicate this across projects. Developers were responsible for risk within the project space.

7.14 Aligning practices with business strategy

A wide scope of business strategies could have featured in this data analysis which would, at best, have culminated in a diluted representation and detraction from the core focus of program management. Instead sights were centred on strategies of self-managed teams, client interaction and involvement, and teaching and coaching of developers. Practices in support of these strategies are summarized below.

Self-managed teams are decreed in most annals of agile software development. However, the notion that a completely self-managed team was a utopian state slowly crystallized. Evidence from observations, which were augmented by the interviews, instead revealed quasi self-managed teams. Organizing and planning of work at the project level was done by the developers. However, management influenced daily work plans in search of harmony in the program management space. Revelations from Scrum practices such as daily stand-up meetings, and artifacts like sprint backlog, influenced these macro-management decisions.

Client interaction did not merely centre on technical communication and software demonstrations. Instead additional practices were instilled to improve trust and to permit

closer engagement between SMME and client. Practices of educating the client on the intricacies of the Scrum process was considered an important venture. Proof of this importance was exaggerated at Minerva where a previous client was coached for a period of almost nine months. Both SMMEs upheld the iron clad strategy not to engage with clients who did not first buy into the Scrum process. The SMMEs could not afford to detract from their tried and tested Scrum processes and allocate scarce resources for investigating alternate internal processes.

Coaching was incorporated into many Scrum practices, including daily stand-up meetings, skills development workshops, pair programming, formal outside training, sprint retrospectives, and informal communication and meetings. Despite the expenditure measured in both time and effort, senior personnel often doubled as mentors instead of carrying out corrective action themselves. The prevailing ethos was one of allowing developers a rite of passage and time to develop with the long-term aim of developing an experienced and resilient workforce.

A disproportionate balance between productivity and learning was observed during projects; disproportionate because most developers demonstrated a natural tendency towards venturing and exploring new development techniques, business terrains and technologies whenever project progress permitted. Otherwise, the limited human resources were mobilized to meet project deadlines. Being technologically inquisitive was one of the recommendations for an SMME employee and most developers demonstrated this characteristic. There were working hours dedicated towards the purpose of improving personnel's technical skills. Whenever a project dipped deeply into the proverbial red, practices such as innovation workshops were held to help better understand causes of problems and work towards resolution. Sprint retrospectives were religiously practiced at Athena.

Table 24: Aligning practices with business strategies antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Aligning practices with business strategies**	**Emergent Nature of Relationship**
Low tolerance for failed projects. Generalist nature of employees required them to deal with peripheral tasks of	Sprint planning sessions and daily stand-up meetings supported the strategy of self-organizing teams.	Empowered teams to self-organize at the project level.	Multi-level decision making but within margins set by management.

project management reporting.	Workshops held for product backlog planning and grooming.		
Shared understanding of the importance of good client relationships and trust. Small clientele and limited marketing, preferred close client relations.	Sprint reviews and onsite client representatives sought to improve client engagement.	Engaged client as an active, contributing member of the project. Helped the client to become well versed in Scrum processes and hence, build trust. Hand-picked clients that already are, or willing to, become proponents of the Scrum movement.	Additional dimension of client engagement whereby the effort expended on client education was seen as an investment.
Lightweight management structure provided an environment conducive to nurturing creativity and innovation.	Daily stand-up meetings provided platforms to disseminate knowledge of more experienced personnel on a daily basis.	Strived to attain sustainable balance between learning and productivity. Coached instead of managed the workforce. Provided developers with space and means to increase their experience and skills.	Coaching and teaching was an integrated practice. Management sought to ensure a mentality of equal balance between learning and productivity.

7.15 Learning practices

How was knowledge transposed from tacit to explicit? Metacognitive theory is the concept of being conscious of how individuals learn and how they aim to improve on that ability (Flavell, 1979, Bråten, 1991). Metacognition refers to reflection with the aim of control over the cognitive processes constituting learning. Stated more simply, it is about *learning about one's learning*. This is a mapped process: "We propose that individuals construct metacognitive theories for two reasons: (a) to systematize their metacognitive knowledge, and (b) to understand and plan their own cognitive activities within a formalized framework" (Schraw and Moshman, 1995). Part of the observed practices in support of learning progressed through the stages of admitting difficulty in action, expending effort to resolve experienced challenges, and establishing and focusing on the cause. Such practices empowered the team to take control of their own learning by being reflective of their actions.

Scrum practices that influenced the developers' skills included pair programming or the buddy system in Minerva's case, informal communication within the open plan office, and developer workshops. Reflective practices included sprint reviews, manager meetings, and sprint retrospectives. Developers were encouraged to provide feedback on strategies, particularly at Athena where this was a mandatory requirement. A conference paper was published during the course of this research highlighting these learning practices; see Singh and Sewchurran (2013).

Due to the close layering of management and developers, project level challenges were easily sighted. Most of the interventions stemming from these reflective practices were implemented immediately. Code reviews were instilled in pursuit of quality in code. Code reviews also afforded a mechanism for standardisation of software products across projects, and unearthed a recurring problem within the code across projects. Electronic aids were in place for organizational learning but these did not appear well sustained in both cases. As a result, most of the organizational learning and knowledge was held tacitly but seemed easily transferred due to the SMME characteristic of close working relationships.

There was dedicated time during office hours for practices to improve skills of the developers. These were set aside and religiously adhered to. The method for transference of knowledge was left entirely up to the developer. There were some instances where the developers reported, at best, questionable effectiveness of this approach. Sprint reviews and sprint retrospectives are embedded within the Scrum process and by default are compulsory practices. In both cases training was also organized with outside parties. However, none of this was observed during fieldwork, although it was mentioned during conversations and interviews.

Overall the process of documenting lessons learned at the program management space was somewhat neglected. Instead, most of this knowledge was held tacitly. The nature of the SMME workforce being a small close-knit community of developers, permitted adequate informal transference of knowledge. Practices such as pair programming and sprint retrospectives allied this phenomenon.

Table 25: Learning Practices antecedents, defining attributes, and consequences

Antecedents		Defining Attributes	Consequence
SMME Characteristics	**Scrum Practices**	**Learning Practices**	**Emergent Nature of Relationship**
Employees were encouraged and given support to improve on their skills and expertise. There was often direct communication mechanisms between management and the workforce. Close and free-flowing interaction between management and employees often ensued.	Code reviews provided a direct feedback loop for measuring quality of work at the project level. Sprint reviews provided a direct feedback loop from clients. Sprint retrospectives provided feedback mechanisms for project teams and for management.	Provided space and means for employees to improve on their individual skills and expertise. Similarly, provided space and means for teams to develop and improve their working relations.	Management of both SMMEs realized the importance of continual scrutiny and improvement of both individual developer and team skills.
Small workforce and relatively flat organizational structure supported informal knowledge transfer. Stringent organizational learning documentation at the program management level seemed obsolete due to dynamic processes, and limited manpower.	Informal communication served as primary conduit for knowledge transfer. Repositories were maintained but had limited use towards organizational learning.	Aimed to improve strategies for improved organizational learning. Encouraged developer feedback on higher level strategic operations. Shared tacit knowledge and experience throughout organization. Continually reviewed and evaluated strategies and internal processes.	Learning practices aimed at continuous evaluation of development processes for efficiency from the perspective of the developers. Organizational learning transfer mostly via informal communication, but this represented an area in need of improvement.

7.16 Quality considerations in the knowledge areas

Quality was not viewed as a separate activity; instead it was integrated into all knowledge areas. There were no explicit stand-alone policies or procedures dedicated towards enforcement of quality. In place of strict quality conformance frameworks every developer, manager and owner was trusted to ensure both high-quality and ethical considerations in their daily tasks. There were instances of observed Scrum software development techniques that

did promote high-quality software production such as software testing being performed by a dedicated developer; as well as pair programming; end of sprint client demonstrations; and use of software architecture frameworks.

7.17 Insights into the stability of observed activity systems

Recall from Chapter Four that a key component of AT epistemology is the notion of the zone of proximal development. In AT, an activity system is perceived as passing through phases of transformation in response to experienced internal contradictions as it strives for stabilization. Traipsing through these transformative phases is referred to as the zone of proximal development. Once the activity system is consistently producing desired outcomes, it is deemed to have reached a state of stability.

Each program management knowledge area was viewed and explored as an activity system, each with its own set of interacting practices and characteristics. Interpretation of the data analysis revealed that an overwhelming majority of these activity systems still exist in a zone of proximal development. Fieldwork revealed that with the exception of infrastructure design, program management knowledge areas and their embodied practices existed in a perpetual state of contradiction; albeit in a long-term cycle of propositioning, testing, evaluation, and change. Despite the short duration of fieldwork during which this nature of practices emerged, the narratives of different personnel show a long serving state of challenges and evolution.

7.18 Summary of chapter

This chapter provided an interpretation of the data collected and analyzed for each case study in Chapter Six. This chapter merged the findings of the two case studies and further refined them via concept analysis concepts of antecedents, defining attributes, and consequences. The resultant interpretation aims to address the research sub-questions why do the characteristics of software SMMEs influence program management practices?, and how do the key activities of Scrum influence program management practices? In reference to the first question, the SMME characteristics were described as antecedents that shaped the program management knowledge area leading to the observed nature of practice. Likewise, Scrum practices also featured as antecedents that honed program management into the observed nature of practice. The conceptual models presented in this chapter depict the nature of the interfaces between Scrum, SMME characteristics and program management practices segregated by program

management knowledge areas. Concept analysis provides a temporal disposition of the interfaces in these models.

In essence, the major findings and the contributions of this study to the existing scholarly body of knowledge are the tabulated theoretical constructs showing the influence of SMME characteristics and Scrum practices on program management. A claim for improved program managment is a bold one, instead this thesis offers a new perspective on the existing body of knowledge on higher order management functions necessary for sustained agile software projects. This is a response to persisting questions like "How can we optimize and how should we apply Scrum agile methods to large-scale and mission critical software development projects?" (Kim, 2007). The theoretical constructs presented in this chapter adds to the growing number of research contributions for instance, Vähäniitty and Rautiainen (2008b), Cho et al. (2011), Pikkarainen et al. (2012) and Van Waardenburg and Van Vliet (2013) who are trying to address the uncertainty and inherent challenges of agile adoption from an organizational management perspective. It achieves this by providing insights of practice along two dimensions of influence namely, SMME characteristics and Scrum practices, for identified program management knowledge area.

The next chapter revisits the research questions and attempts answers to these questions, identifies limitations of the study, and suggests possible areas for future research.

CHAPTER 8: CONCLUSION, LIMITATIONS AND FUTURE RESEARCH

8.1 Introduction

This chapter concludes the thesis. It presents the findings of the study in relation to the aim, which is to theorize a perspective of practices in the multi-project space of agile software development. This aim was strategized and operationalized through the three main objectives of the thesis which collectively sought to uncover the interfaces between program management practices, SMME characteristics, and Scrum practices. In this quest the thesis adopted an interpretive perspective, used abductive reasoning for propositioning and inductive reasoning for theory formulation, and employed multi-case study sites. The case study methodology best portrayed the contemporary descriptive style. Activity Theory (AT) formed the data analysis lens and concept analysis helped to shape the interpretation of analysis and to present a view of interfaces.

In the rest of the chapter an overview of the research is provided; the research questions are revisited; the intended audience of this thesis is described; and the research contributions are presented. Limitations of the study are then presented. The chapter concludes with suggestions for future research.

8.2 Overview of the research

Chapter One established the context of the study and made a case for its meeting a persistent need. Scrum is an organizing approach for software development with a growing mass of followers and contributors. Popularized via copious accounts of implementation and related matters of pragmatic application, Scrum has recently proliferated within the project level space. Nonetheless, persistent questions surround the impact and influence of this agile software development approach on enterprise and program level operations of organizations. Furthermore, Scrum and its implications on higher order internal organizational processes such as program management, are largely missing from academic scrutiny and debate. This thesis posits that the inclusion of Scrum in theoretically biased academic research may serve the long-term enterprise of increased understanding and improved robustness in Scrum's organizational adoption and integration. This thesis sought to make a theoretical contribution in this context. To the best of the researcher's knowledge, no study has set out to better

understand program management practices as they influence, and are influenced by, both SMME characteristics and Scrum practices.

The three core areas of interest, namely program management, software development SMMEs, and Scrum software development, were introduced in Chapter One. Each represented a vast area of enquiry and the presentation of reviewed literature in Chapter Two aimed to reconcile and focus the research effort. As both case study sites were software SMMEs, the impact of the nature of their management practices and characteristics on program management had to be carefully considered. The literature review sought to define SMMEs and to portray the nature of their management practices and characteristics. Agile software development was then detailed together with a perspective on its underpinning guidelines. As a popular derivative of agile, Scrum software development and practices were described in detail along with trends and state-of-the-art in its adoption and implementation. Program management was defined. A synthesis of knowledge areas and practices was uncovered to provide clues for the fieldwork with the proviso that emergent practices should be anticipated and taken into consideration. The view adopted in this thesis was that the three core areas both influence and are influenced by one other, but it is important to take cognisance of where they have also formed a symbiotic co-existence. What does this interface look like? This was the key question driving this study, and thus the overlapping or intersection of the three core areas formed the theatre of operations for this study.

A suitable lens was sought to investigate and unearth this interface between program management, SMME characteristics, and Scrum practices. Work Systems Method, Soft Systems Methodology, and Actor-Network Theory were early contenders during this search. However, the epistemology and concepts of Activity Theory (AT) embodied the most suitable mechanisms for appreciating the socio-technical latticework and dynamic nature of the observed internal processes. The major contributing actors and change agents included a human contingent consisting of clients, owners, managers and developers, as well as non-human actors like process methodologies, strategies, best practices and a wide range of technical artifacts and support tools.

Chapter Three described the research methodology (with the exception of the data analysis, which was covered in Chapter Four). A qualitative design was appropriate due to a small sample size with prolonged fieldwork at each site. In attempting to understand the intricate interfaces, varying perceptions and opinions of software developers and project teams had to be distilled. As a result, the interpretive research stance played a crucial underpinning role for

data collection and analysis. The chapter explained the choice of descriptive as the case study strategy, and the associated data collection techniques, that comprised mainly observation, interviews, and artifact analysis. Both proved adequate for this research undertaking.

Key concepts of AT that were utilized in the data analysis, were discussed separately in Chapter Four, along with some insights of how other studies applied Activity Theory in achieving their research goals.

The thesis then progressed, with more in-depth detail provided on the two case study sites in Chapter Five. By employing pseudonyms, the identities of the software SMMEs and personnel were protected. Personnel organograms were illustrated and brief histories of the SMMEs presented. The level of detailed description evident in Chapter Five was recommended due to the small sample size and the long and intimate fieldwork.

AT formed the major epistemological lens that shaped the data analysis. The concepts of mediating actors within a conceptual purposeful activity system were sufficient for unearthing and making explicit the observed interactions between people, organization, and methodology. These were presented in Chapter Six. The key concept of mediating actors was helpful in gaining an understanding of the forces influencing the trinity of this thesis, namely program management, software SMMEs, and Scrum. Further, AT did not distinguish between human and non-human actors within the system. A major difference between this and other studies that implemented AT resided with data collection and analysis being conceptualized as multiple activity systems; that is, one activity system per program management knowledge area. This was necessary due to the complexity revealed in relationships between unearthed program management, SMME characteristics and Scrum.

The study treats each case study as a separate unit of analysis to determine their individual program management interfaces with Scrum and SMME characteristics. The theoretical constructs, namely AT representation of program management knowledge areas, were completed for each SMME. In Chapter Seven an interpretation of the data analysis was done. Concept analysis provided a basis for the interpretation framework which allowed for a temporal dimension in the findings. The theoretical constructs produced at that stage were the tables representing the interfaces between SMME characteristics, Scrum practices and Program management, per knowledge area. Theoretical constructs where shaped by Llewelyn's (2003) proposal for theoretical levels, while the type of theory being developed conformed with the explanation category of Gregor's (2006) taxonomy of theories. Despite

each case study SMME having a unique modus operandi, there were significant similarities due to SMME characteristics and Scrum practices being mostly of the generic variety. This study did not set out to generalize findings across a large sample size.

8.3 Revisiting the research questions

The success of the case study SMMEs, lay in their ability to harmonize the conflicting areas of internal business processes, SMME traits, and the Scrum development framework. The contention of this thesis was that program management practices are influenced by SMME characteristics and Scrum practice, and that the emergent program management practices would differ to some degree from a priori program management practices. So, what do these interfaces look like? How do they influence and how are they influenced by program management practices?

To the best of the author's knowledge, no similar, comprehensive study has been undertaken to date to demystify these uncertainties in the context of agile software development environments. The analysis of data from the two SMME case sites revealed that their economic viability and success rate could be attributed to their ability to temper program management into practices which were mostly fluid and reactive in nature. The characteristics of SMME and Scrum practices played both supportive and restrictive roles. The findings of this research endeavour were arrived at by answering the primary and the sub research questions. The primary research question was as follows:

MAIN RESEARCH QUESTION:

> *How is program management carried out in software development SMMEs practicing Scrum?*

The purpose of answering the primary research question was to gain a better understanding of the nature of program management practices within the context of an agile software development environment. Program management practices were discovered to be strongly connected, and found to be swayed by both the characteristics of SMMEs and by the practices of the adopted agile approach, in this case Scrum.

Ultimately, five subquestions served to structure the study in finer grain; that is, to search for the reasons as to how SMME characteristics and Scrum forced the evolution of the observed, evolved, program management. There are more subquestions listed here than in Chapter

Three. This is in keeping with a descriptive case study where questions clarify and emerge during the progression of the lengthy research endeavour.

SUB RESEARCH QUESTIONS:

1. *What are the characteristics of software SMMEs?*
2. *What are the key practices in Scrum?*
3. *What are the key knowledge areas for program management in software development environments?*
4. *How do the characteristics of software SMMEs influence program management practices?*
5. *How do the key activities of Scrum influence program management practices?*

Research subquestions 1, 2, and 3 were investigated via desktop research and findings were presented in Chapter Three. Questions 4 and 5 required more attention. Each is elaborated on individually below:

> *Sub Research Question 4: Why do the characteristics of software SMMEs influence program management practices?*

This question helped to better understand how the SMME characteristics influenced program management practices. This question featured in Chapters Two, Six, and Seven. Chapter Two provided initial insights into the unique characteristics of software SMMEs, but it must be stressed that current literature lacked emphasis on or insight into South African software SMMEs. Chapters Six and Seven analyzed and interpreted data respectively, and this process unravelled SMME characteristics that were discovered to be mostly aligned with those found in the literature study. The analysis and interpretation process also revealed new characteristics.

> *Sub Research Question 5: How do the key activities of Scrum influence program management practices?*

This question sought to expose the impact of Scrum practices on program management. Like the sub question above, this question featured in Chapters Two, Six, and Seven. In Chapter Two a discussion ensued on current perception of Scrum as a software development framework, as well as exploration of Scrum practices and guidelines. Although Scrum is considered the *new kid on the block* in the company of more established software development approaches, both its framework and its practices appeared well established among both pragmatists and academia, with little variation in implementation at the

individual project level. The data analysis and interpretation in Chapters Six and Seven respectively, revealed few differences between a priori and the case study implementations of Scrum.

8.4 The intended audience of this study

Before writing up the findings of this study consideration was given to who the intended audience will be (as advised by Yin (2012) and Creswell (2013)). Following their guidelines, the intended audience of this thesis was stated to be as follows: first, the examiners charged with grading this thesis; second, the participants from the SMME case sites; and third, the agile research community. Given these variances in audience, the case study report had to demonstrate methodological and theoretical integrity, as well as provide at least some pragmatically useful insights for both the SMMEs that volunteered participation and the larger agile research community. Attempts were made to address these disparate needs in this study.

8.5 Theoretical research contributions

To the best of the researcher's knowledge, this is the first comprehensive study that seeks to demystify the interfaces between program management practices, software SMMEs and the Scrum development framework. This research has culminated in findings that are believed to be important in understanding the nature of the SMME business environment and the internal business practices that have the potential to support or hinder Scrum. This notion is expanded by contentions made by Krishnan and Ulrich (2001), who proclaim that the popular how-to-succeed recipes are mostly temporal and differ across SMMEs, but some of the essential ingredients remain. The major theoretical research contributions of this study are described in the following subsections.

8.5.1 Interface between program management practices, SMME characteristics, and Scrum practices

This study's mainstay contribution lies in the conceptualization and representation of program management knowledge areas and embodied activities within the Scrum software development discourse. These knowledge areas and activities could not be accurately portrayed in isolation. Instead, this thesis theorized a set of interactions between program management, SMME characteristics, and Scrum practices. More specifically, it sought to unearth how SMME characteristics and Scrum practices influence the program management

practices. The mediating relationships between these central areas for each case were explored in Chapter Six and the major identified influences were conceptualized in Chapter Seven. Figure 15 below illustrates these mediating relationships:

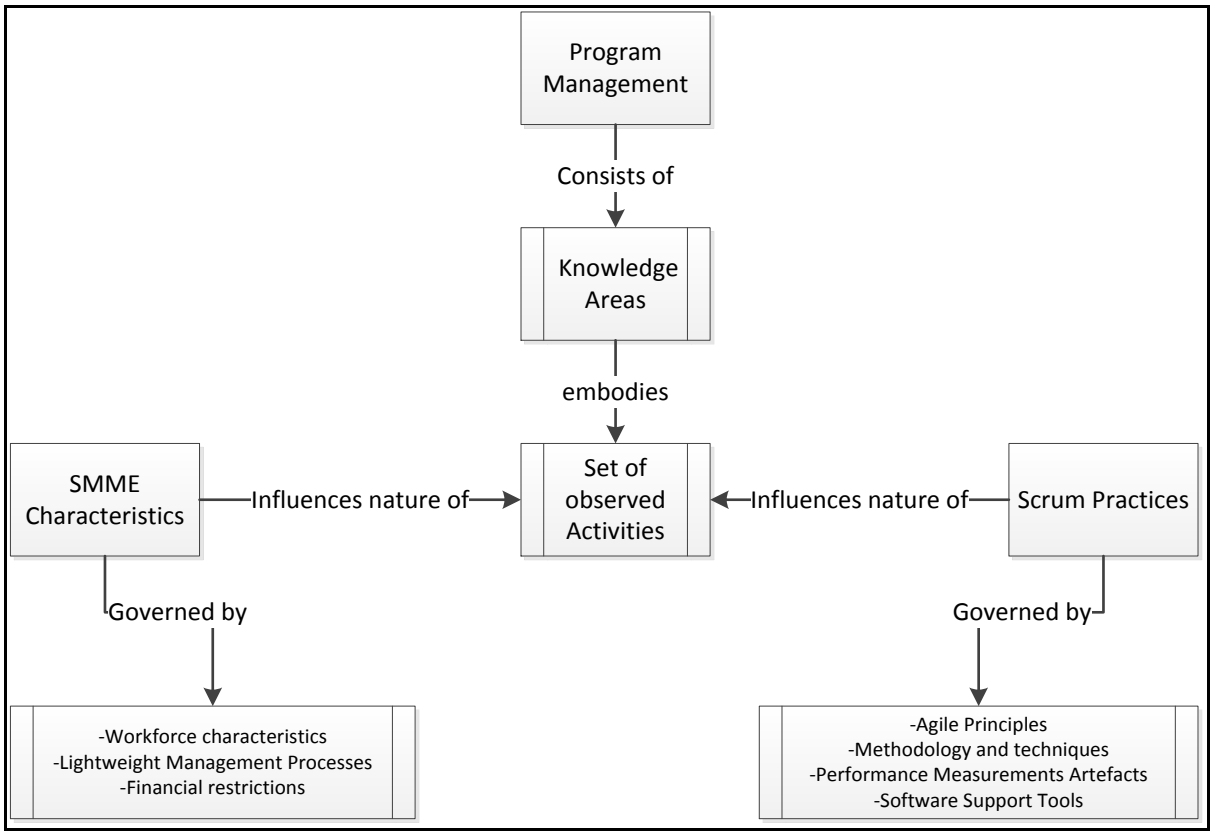


Figure 15: Collaboration Diagram highlighting the influences on Program Management. Source: Researcher's own construction.

8.5.2 Nature of emergent program management practices in context of agile software development

The majority of observed program management practices differed in nature to some extent from the more traditional software engineering management frameworks explored in Chapter Two. This study found that strict economic project focus, strict command and control mentality, and conformance to standards, seem to have mostly been conceded to more fluid, reactive, and adaptive management practices. Underpinning most of the observed program management practices was a bottom up organizing approach and not just a strict top down flow of command. The program management practices at both case sites were in a state of evolution and development for the entire duration of fieldwork. However, the practices in some cases were similar to those identified in existing literature such as, Dingsøyr et al. (2006), Cho (2010), and Usman et al. (2014). Although Scrum was successfully managed by

individual teams at the project level, program management practices appeared in different states of proximal development.

Overall, program management can be seen as a complex adaptive system which requires altered tools and practices when compared to the ones derived from a more deterministic and pure *iron triangle* management perspective. These tools and practices are provided and shaped by the interfaces between SMME characteristics and Scrum practices.

8.5.3 Usefulness of Activity Theory as a lens for exploring mediated program management practices

The literature review highlighted a few project management frameworks. Despite their popularity, this study posits that these frameworks were not well suited lenses for deep exploration of an agile software project environment. The observed environment manifested as a set of complex socio-technical systems which allied both human and non-human actors. Actors included people, methodologies, innovation, ideas, software and communications technology, and a range of automated software support tools. Using the concept of an activity system, Activity Theory allowed for a description of this liquefied socio-technical network. Program management practices in the context of a Scrum software environment did not surface as a neat, linear sequence of activities, but was more akin to chains of interactions, growing in size and nature as projects progress. AT further allowed for the discovery of inhibiting and supportive influences on practices, due to its epistemology of mediating components. It is thus the view of this thesis that AT was a suitable lens to identify and interrogate program management practices within a highly dynamic agile software environment. This was the focus of a conference proceeding published during the course of the study. AT also revealed that most of the observed program management practices haven't yet reached a high level of stabilization and most still appeared to be passing through the zone of proximal development.

8.5.4 Usefulness of Concept Analysis

The entire transformation of the program management knowledge area or activity system could not be observed due to the short duration of fieldwork. Instead, a start and end point within the transformation is documented. This may be referred to as a *snapshot* of the activity system passing through the zone of proximal development. Concept analysis was uncovered as a suitable framework to capture this temporal dimension. Using a model of concept analysis proposed in Johns (1996), a framework was developed to showcase the relationships

between SMME characteristics and Scrum practices within each program management knowledge area. Using epistemology of antecedents, defining attributes and consequences, concept analysis was found to be a useful aid and catered for the missing temporal dimension in highly dynamic practices at the program management level.

8.6 Practical research contributions

How does this thesis make a contribution towards the pragmatic and academic areas of agile software development program management? In seeking the answers to the research questions, certain traits and characteristics of program management practices were exposed. In being made explicit in this study, these might contribute towards advancement of current understanding of the nature of the SMME business environment and program management practices necessary for the germination of effective agile software development approaches such as Scrum. The subsections that follow look at the study's major practical contributions towards realizing this objective.

8.6.1 Balancing of productivity and learning

Both case study companies invested significant effort and expenditure on learning practices. It was not uncommon for software production to come to a halt on an underperforming project in order to make room for reflection and learning. Scrum permitted incorporation of learning practices into daily routines, while small and close camaraderie of the SMME workforce ensured learning; and lessons learnt were disseminated with as little effort as possible. One significant downfall was the lack of documentation of lessons learnt and the potential problem of having to *reinvent the wheel* in this regard. The small workforce carried most of this knowledge tacitly and transference appeared satisfactory.

8.6.2 Quasi self-organizing teams

The agile mantra of self-organizing teams resonated for both case sites at the project level. However, the Scrum Master played the key role of orchestrator and regulated these self-organizing teams to achieve harmony at the program management level. The individual teams accepted the disruptions to their project level work plans caused by the Scrum Master, mostly because they are conditioned to do so, and because they accepted the nature of fluidity and dynamism inherent in SMME management processes. Developers sometimes hand-picked their project assignment and many worked on more than one project at a time. The SMME generalist nature of an employee, coupled with multi-project awareness facilitated by Scrum

tools, and informal arrangement, catered for this cross-project movement. Developers often picked projects that challenged their ability, but the Scrum Master enforced allocations with positive project progress in mind. Developers often portrayed a strong sense of ownership over an assigned project and lobbied for additional resources when the need arose.

8.6.3 Multi-level decision making

Program management decision-making was carried out at multiple levels. At the project level, release planning, daily task allocation, and estimation was achieved by developers via a self-organizing approach. Interruptions to the project level stemmed mainly from the decisions made at the program management level, the latter being caused by the continuous search for ways of harmonizing the milestones and ensuring progression of the multiple concurrent projects. Usually this was a reactive response to changes in project parameters such as scope creep and shortage of developers. The Scrum Master was charged with this duty of multi-project harmonizing.

8.6.4 Dynamic practices

Even though it was hinted at during the course of the literature review, the observed level of dynamism in practices was surprising. Heavily allied by software support tools, up-to-the-minute reporting was possible and the associated adjustments to work plans could be made with immediate effect. Upon request, clients were instantly provided with progress reports. In both cases, the Scrum Masters reported that the processes for cost estimation and scheduling were not adequate and were under constant evolution. Lightweight management processes, which is a common characteristic of SMMEs, coupled with the short delivery cycles of Scrum, provided an ambit for this level of dynamism. Does it work? Yes, but not without intense coordination from the Scrum Master.

8.6.5 Constant review and adjustment of program management

Bedded firmly in the Lean philosophy of perfection being an ideal, frequent modifications to most program management practices were observed. Non-rigid structures of the SMMEs, coupled with Scrum being more of a framework of guidelines rather than a rigid prescriptive structure, catered for such frequent alterations. Strategic level adjustments take a longer time before their impact is empirically noticed. However, these were from time to time scrutinized by employees and they had the opportunity to provide input into the long-term, strategic direction of the SMME.

8.6.6 Client engagement

Scrum guidelines suggest that barriers between development team and client should be minimal. The flow of communication between client and developer team was mostly mediated by the Scrum Master. The on-site client liaison was sparsely observed in projects. There was usually a large extent of effort that went into inducting the client into the ways of Scrum. Clients were educated on generic Scrum procedure and ceremony as well as in its tailored form within the SMME. In both cases this educating process was deemed a necessity in support of nurturing strong client relations and trust. Furthermore, the client was responsible for the early requirements phase via the role of Product Owner. Clients that were not supportive of Scrum were typically not engaged.

8.7 Research limitations

The first limitation of this study is the restriction to two SMME case sites. The study would have been enhanced with more SMME case study sites with slight variations in their Scrum implementations, as this would have allowed for a greater grasp of understanding of program management practices. The study also focused on SMMEs and did not study medium and large companies, or in-house software development environments. Investigations within these contexts would have widened the understanding of program management practices. The SMMEs used in this study were the only two that obliged as volunteers out of eight requests that were made.

In this study, knowledge areas within the discourse of program management totalled eleven. This prevented a high degree of detail, and reduced the ability to drill down and allow for more concise exploration and reporting within each area. However, identification of both existing and emerging knowledge areas was seen as an important first step. This hopefully creates new terrains for more detailed and more in depth investigations.

The duration of the fieldwork could be considered a handicap. By the end of the fieldwork period in both SMMEs, very few new instances of data were being documented but new data had not entirely ceased. This is aligned with the yet to be settled nature of program management practices. An even longer duration of fieldwork of perhaps one year within each SMME, would have been ideal. This was not permitted due to the researcher’s full-time employment responsibilities and the lack of willingness by the SMMEs due to potential disruption.

8.8 Future research

Several areas of future research where uncovered during the course of this study. The following paragraphs flesh out these possible avenues for new research.

This study should be expanded to larger software development companies. With a small workforce and limited number of projects this study was able to ring-fence program management practices. However, the design of program management within larger software companies which have multiple layers of hierarchy and many concurrent projects, is relatively unknown. The knowledge areas uncovered in this thesis might provide an adequate lens into these contexts.

The case study SMMEs were very selective of their clientele. Only those clients that believed in or were willing to convert to agile, and in particular to Scrum, were entered into with software contracts. The question that arises is: What is the nature of practices when a contradiction occurs between a client that subscribes to classic project management frameworks (like PMI), and a software development team practicing agile? Furthermore, in the case of in-house software teams, how is integration done between the traditional management framework within the enterprise level, and agile software development at the operations level?

Cost estimation is an area of agile that is largely unsettled. This is evident in both case study sites and is mostly due to constant changes to project parameters, such as requirements changes, technology discovery, and change to project personnel. Modelling techniques need to be investigated in an attempt to provide potential reprieve in this area.

The notion of tacit organizational learning within the Scrum development environment was intriguing. Although one paper was published laying down some grounding, much more elaborate investigative undertakings should focus on the transference of knowledge, expertise and experience within the program management space.

There is an alarming shortage of published statistics with regards to South African software SMMEs. This problem stems in part from “software” not being a classification that is generated within primary South African statistics gathering on organizations (such as that by the South African Revenue Service). As a result software companies can choose to self-classify under “business services” or within the sector in which the do most of their development (such as “finance” or “tourism”). Furthermore, due to the cross-cutting nature of the IT sector, the Sectoral Education Authority for ICT, the MICT SETA, also does not have

required data and also does not have “software” as a classification. There have been efforts to research the software sector in certain regions, such as the Western Cape, but these studies are now outdated or mostly qualitative. There is a dire need for quantitative national data on software SMME characteristics, company profiles, development methodologies adopted and economic turnover. Project success rates and reasons for success and failure would also contribute towards creating a perspective of the South African software SMME landscape.

Finally, a quantitative study could be commissioned to model the documented program management practices to determine the frequency of occurrence and thus further confirm the findings of this study. This should be done on a wider scale, and include international software SMMEs.

References

ABRAHAMSSON, P., CONBOY, K. & WANG, X. 2009. "Lots done, more to do": the current state of agile systems development research.

ABRAHAMSSON, P., WARSTA, J., SIPONEN, M. T. & RONKAINEN, J. 2003. New Directions on Agile Methods: A Comparative Analysis. *International Conference on Software Engineering.* Portland, USA: IEEE.

AGILEMANIFESTO. 2001. *Manifesto for Agile Software Development* [Online]. Available: http://agilemanifesto.org/. Viewed 04 June 2009.

AHLEMANN, F. 2009. Towards a conceptual reference model for project management information systems. *International Journal of Project Management,* 27**,** 19-30.

AHLEMANN, F., EL ARBI, F., KAISER, M. G. & HECK, A. 2013. A process framework for theoretically grounded prescriptive research in the project management field. *International Journal of Project Management,* 31**,** 43-56.

ALAJOUTSIJÄRVI, K., MANNERMAA, K. & TIKKANEN, H. 2000. Customer relationships and the small software firm: A framework for understanding challenges faced in marketing. *Information & Management,* 37**,** 153-159.

ALLEN, D. K., BROWN, A., KARANASIOS, S. & NORMAN, A. 2013. How Should Technology-Mediated Organizational Change be Explained? A Comparison of the Contributions of Critical Realism and Activity Theory. *MIS Quarterly,* 37**,** 835-854.

AMBLER, S. W. 2002. Lessons in Agility from Internet-based Development. *Software, IEEE,* 19**,** 66-73.

AMBLER, S. W. 2008. *Agile adoption rate survey results: February 2008* [Online]. Available: www.ambysoft.com/surveys/agileFebruary2008.html. Viewed 09 November 2010.

AMBLER, S. W. 2009. Scaling Agile Software Development Through Lean Governance. *International Conference on Software Engineering.* Vancouver, Canada: IEEE.

APMG-INTERNATIONAL. 2012. *APMG-International Accrediting Professionals* [Online]. Available: http://www.apmg-international.com/en/qualifications/prince2/prince2.aspx. Viewed 02 March 2012.

ARCHER, M. 1998. Introduction: Realism in Social Research. *In:* ARCHER, M., BHASKAR, R. & COLLIER, A. N. (eds.) *Critical Realism: Essential Readings.* London. UK: Routledge.

ARELL, R., COLDEWAY, J., GATT, I. & HESSELBERG, J. 2012. *Characteristics of Agile Organizations* [Online]. Available: http://www.agilealliance.org. Viewed 10 August 2013.

ARNOLD, E. 2006. *Facilitating university sustainability through decision-orientated financial reporting.* Master of Education (Policy Analysis, Leadership and Management) Mini-Thesis, University of the Western Cape.

ARTTO, K., MARTINSUO, M., GEMÜNDEN, H. G. & MURTOARO, J. 2009. Foundations of program management: a bibliometric view. *International Journal of Project Management,* 27**,** 1-18.

ÅSVOLL, H. 2013. Abduction, deduction and induction: can these concepts be used for an understanding of methodological processes in interpretative case studies? *International Journal of Qualitative Studies in Education***,** 1-19.

ATKINSON, R. 1999. Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International journal of project management,* 17**,** 337-342.

AUBRY, M., HOBBS, B. & THUILLIER, D. 2007. A new framework for understanding organisational project management through the PMO. *International journal of project management,* 25**,** 328-336.

AUBRY, M., HOBBS, B. & THUILLIER, D. 2009. The contribution of the project management office to organisational performance. *International Journal of Managing Projects in Business,* 2**,** 141-148.

AUGUSTINE, S., PAYNE B., SENCINDIVER, F. & WOODCOCK, S. 2005. Agile Project Management: Steering from the Edges. *Communications of the ACM,* 48**,** 85-89.

AZAR, J., SMITH, R. K. & CORDES, D. 2007. Value-oriented requirements prioritization in a small development organization. *Software, IEEE,* 24**,** 32-37.

BABB, J. S. & NØRBJERG, J. A Model for Reflective Learning in Small Shop Agile Development. *In:* MOLKA-DANIELSEN, J., ed. Information Systems Research Seminar 20th -22nd August, 2010 2010 Rebild, Denmark 23-38.

BARKER, S. 2013. *Brilliant PRINCE2: What you really need to know about PRINCE2,* Harlow, UK, Pearson Education Limited.

BARTON, B. 2012. *Agile Has Crossed the Chasm, Agile Hasn't Crossed the Chasm* [Online]. Cutter Blog. Available: http://blog.cutter.com/2012/07/31/agile-has-crossed-the-chasm-agile-hasnt-crossed-the-chasm/. Viewed 08 October 2012.

BASKERVILLE, R. L., PRIES-HEJE, J. & MADSEN, S. 2010. From Exotic to Mainstream: A 10-year odyssey from Internet Speed to Boundary Spanning with Scrum. *In:* DINGSOYR, T., DYBA, T. & MOE, N. B. (eds.) *Agile Software Development. Current Research and Future Directions.* Berlin: Springer.

BECK, K. 1999. Embracing Change with Extreme Programming. *Computer, IEEE,* 32**,** 70-77.

BEEDLE, M., DEVOS, M., SHARON, Y., SCHWABER, K. & SUTHERLAND, J. 1999. SCRUM: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design,* 4**,** 637-651.

BENBASAT, I., GOLDSTEIN, D. K. & MEAD, M. 1987. The Case Research Strategy in Studies of Information Systems. *MIS Quarterly,* 11**,** 369-386.

BENEFIELD, G. Rolling out agile in a large enterprise. Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 7-10 January 2008 2008. Waikoloa, Big Island, Hawaii, USA: IEEE.

BERRY, A., VON BLOTTNITZ, M., CASSIM, R., KESPER, A., RAJARATNAM, B. & VAN SEVENTER, D. E. 2002. The economics of SMMES in South Africa. *Trade and Industrial Policy Strategies,* 1.

BHASKAR, R. 1978. *A Realist Theory of Science,* Sussex, Harvester Press Limited.

BHASKAR, R. 1998a. Philosophy and Scientific Realism. *In:* ARCHER, M., BHASKAR, R., COLLIER, A., LAWSON, T. & NORRIE, A. (eds.) *Critical Realism. Essential Readings.* Oxon: Routledge.

BLOMQUIST, T. & MÜLLER, R. 2006. Practices, Roles, and Resposibilities of Middle Managers in Program and Portfolio Management. *Project Management Journal,* 37**,** 52-66.

BOEHM, B. 2002. Get Ready for Agile Methods, With Care. *Computer, IEEE,* 35**,** 64-69.

BOEHM, B. & TURNER, R. 2005. Management Challenges for Implementing Agile Processes in Traditional Developing Organizations. *IEEE Software,* 5**,** 30-39.

BOSCH, J., OLSSON, H. H., BJÖRK, J. & LJUNGBLAD, J. 2013. The early stage software startup development model: A framework for operationalizing lean principles in software startups. *In:* FITZGERALD, B., CONBOY, K., POWER, K., VALERDI, R., MORGAN, L. & STOL, K. J. (eds.) *Lean Enterprise Software and Systems.* Berlin, Germany: Springer.

BRÅTEN, I. 1991. Vygotsky as precursor to metacognitive theory: The concept of metacognition and its roots. *Scandinavian Journal of Educational Research,* 35**,** 179-192.

BRAUDE, E. J. & BERNSTEIN, M. E. 2011. *Software Engineering: Modern Approaches,* Hoboken, New Jersey, USA, Wiley.

BROWN, T. J. 2008. *The Handbook of Program Management,* New York, USA, McGraw Hill.

BURCH, R. 2010. Charles Sanders Peirce. *Stanford Encyclopeida of Philosophy.* Stanford University: Metaphysics Research Lab.

BURRELL, G. & MORGAN, G. 1979. *Sociological Paradigms and Organizational Analysis. Elements of the Sociology of Corporate Life.,* London, Heinemann Educational Books Ltd.

BURTON-JONES, A. & GALLIVAN, M. J. 2007. Toward a deeper understanding of system usage in organizations: a multilevel perspective. *Mis Quarterly,* 31**,** 657-679.

BURTON-JONES, A., MCLEAN, E. R. & MONOD, E. 2011. On approaches to building theories: Process, variance and systems. British Columbia, Canada: Sauder School of Business, UBC.

CARLSSON, S. A. Toward an Information Systems Design Research Framework: A Critical Realist Perspective. DESRIST, 2006 Claremont, CA. CGU.

CARLSSON, S. A. Critical Realist Information Systems Research in Action. *In:* CHIASSON, M., HENFRIDSSON, O., KARSTEN, H. & DEGROSS, J. I., eds. IFIP WG 8.2 Working Conference, June 6-18 2011 Turku, Finland. Springer, 269-284.

CERVONE, H. F. 2010. Understanding agile project management methods using Scrum. *OCLC Systems and Services: International Digital Library Perspectives,* 27**,** 18-22.

CESCHI, M., SILLITTI, A., SUCCI, G. & DE PANFILIS, S. 2005. Project management in plan-based and agile companies. *Software, IEEE,* 22**,** 21-27.

CHAN, F. K. Y. & THONG, J. Y. L. 2009. Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems,* 46**,** 803-814.

CHEN, C. C., LAW, C. & YANG, S. C. 2009. Managing ERP implementation failure: a project management perspective. *IEEE Transactions on Engineering Management,* 56**,** 157-170.

CHO, J., HUFF, R. A. & OLSEN, D. 2011. Management Guidelines for Scrum Agile Software Development Process. *Issues in Information Systems,* 12**,** 213-223.

CHO, J. J. 2010. *An exploratory study on issues and challenges of agile software development with scrum.* Doctor of Philosphy, Utah State University.

COCKBURN, A. 2002. Learning from Agile Software Development - Part One. *Crosstalk. The Journal of Defense Software Engineering***,** 10-14.

COCKBURN, A. 2007. *Agile Software Development. The Cooperative Game.,* Boston, Pearson.

COHN, M. & FORD, D. 2003. Introducing an agile process to an organization. *Computer,* 36**,** 74-78.

COLEMAN, G. & O'CONNOR, R. 2008. Investigating software process in practice: A grounded theory perspective. *Journal of Systems and Software,* 81**,** 772-784.

CONBOY, K. 2009. Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research,* 20**,** 329-354.

CONBOY, K., WANG, X. & FITZGERALD, B. Creativity in Agile Systems Development: A Literature Review. Information Systems - Creativity and Innovation in Small and Medium-Sized Enterprises, IFIP WG8.2 International Conference, 21-24 June 2009 Guimaraes, Portugal. 122-134.

CORAM, M. & BOHNER, S. The impact of agile methods on software project management. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 4-7 April 2005 2005 Greenbelt, Maryland, USA. IEEE, 363-370.

COTTMEYER, M. 2009. *The Agile Project Manager* [Online]. Available: http://www.donnaareed.com/wp-content/uploads/2010/01/TheAgileProjectManager-VersionOne.pdf. Viewed 5 November 2013.

CRAWFORD, K. & HASAN, H. 2006. Demonstrations of the activity theory framework for research in information systems. *Australasian Journal of Information Systems,* 13**,** 49-67.

CRAWFORD, L., MORRIS, P., THOMAS, J. & WINTER, M. 2006. Practitioner development: From trained technicians to reflective practitioners. *International Journal of Project Management,* 24**,** 722-733.

CRESWELL, J. W. 2009. *Research Design. Qualitative, Quantitative and Mixed Methods Approaches,* Los Angeles, Sage.

CRESWELL, J. W. 2013. *Qualitative Inquiry and Research Design: Choosing among five approaches,* California, USA, Sage.

DEETZ, S. 1996. Describing Differences in Approaches to Organizational Science: Rethinking Burrell and Morgan and Their Legacy. *Organizational Science,* 7.

DENNING, S. 2009. *Radical Management* [Online]. Available: http://www.stevedenning.com/Radical-Management/default.aspx. Viewed 23 January 2012.

DENNING, S. 2010a. A leader's guide to radical management of continuous innovation. *Strategy and Leadership,* 38**,** 11-16.

DENNING, S. 2010b. *Leader's Guide to Radical Management: Reinventing the Workplace for the 21st Century,* Hoboken, NJ, Jossey-Bass.

DENZIN, N. K. 2002. The Interpretive Process. *In:* HUBERMAN, A. M., MILES, R. & MATTHEW, B. (eds.) *The Qualitative Researcher's Companion.* California: Sage Publications.

DENZIN, N. K. & LINCOLN, Y. S. 2008. Introduction: The Discipline and Practice of Qualitative Research. *In:* DENZIN, N. K. & LINCOLN, Y. S. (eds.) *Strategies of Qualitative Inquiry.* 3rd ed. Los Angeles: Sage.

DIEFENBACH, T. 2009. Are case studies more than sophisticated storytelling?: Methodological problems of qualitative empirical research mainly based on semi-structured interviews. *Quality & Quantity,* 43**,** 875-894.

DINGSØYR, T., HANSSEN, G. K., DYBÅ, T., ANKER, G. & NYGAARD, J. O. Developing Software with Scrum in a Small Cross-Organizational Project. *In:* MESSNARZ, R., ed. EuroSPI 2006, 2006 Berlin Heidelberg, Germany. Springer-Verlag, 5-15.

DINGSØYRA, T., NERUR, S. & BALIJEPALLY, V. 2012. A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems and Software,* 85**,** 1213-1221.

DOBSON, P. J. 2002. Critical Realism and Information Systems Research: Why bother with philosophy? *Information Research,* 7**,** 1-15.

DOUVEN, I. 2011. *Peirce on Abduction* [Online]. Stanford University: Metaphysics Research Lab. Available: http://plato.stanford.edu/entries/abduction/peirce.html. Viewed 29 September 2013.

DOUVEN, I. 2011b. Abduction. *Stanford Encyclopedia of Philosophy.* Standford University: Metaphysics Research Lab.

DTI 2008. Annual Review of Small Businesses in South Africa 2005-2007. *In:* INDUSTRY, D. O. T. A. (ed.). Pretoria: Department of Trade and Industry.

DYBA, T. 2005. An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering,* 31**,** 410-424.

DYBÅ, T. & DINGSØYR, T. 2008. Empirical studies of agile software development: A systematic review. *Information and software technology,* 50**,** 833-859.

EISENHARDT, K. M. 1989. Building theories from case study research. *Academy of management review,* 14**,** 532-550.

ENGESTRÖM, Y. 1987. *Learning by expanding. An activity-theoretical approach to developmental research,* Helsinki, Orienta-Konsultit.

ENGESTRÖM, Y. 1993. Developmental studies of work as a testbench of activity theory: The case of primary care medical practice. *In:* CHAIKLIN, S. & LAVE, J. (eds.) *Understanding practice: Perspectives on activity and context.* Cambridge, UK: Cambridge University Press.

ENGESTRÖM, Y. 1999. Expansive visibilization of work: An activity-theoretical perspective. *Computer Supported Cooperative Work (CSCW),* 8**,** 63-93.

ENGESTRÖM, Y. 2000. Activity theory as a framework for analyzing and redesigning work. *Ergonomics,* 43**,** 960-974.

ENGESTRÖM, Y. 2001. Expansive learning at work: Toward an activity theoretical reconceptualization. *Journal of education and work,* 14**,** 133-156.

ENGESTRÖM, Y. 2008. *From Teams to Knots. Activity-Theoretical Studies of Collaboration and Learning at Work,* Cambridge, UK, Cambridge University Press.

ENGESTRÖM, Y. 2009. The future of activity theory: A rough draft. *In:* SANNINO, A., DANIELS, H. & GUITIERREZ, K. D. (eds.) *Learning and expanding with activity theory.* Cambridge, UK: Cambridge University Press.

ESPINOSA, D., CROASDELL, D., EDBERG, D. & THORN, L. The New IT Product/Project Lifecycle. Fifteenth Americas Conference on Information Systems, August 6th-9th 2009 San Francisco. 1-14.

FAIRLEY, R. 2009. *Managing and Leading Software Projects,* Hoboken, John Wiley and Sons Inc.

FAYAD, M. E., LAITINEN, M. & WARD, R. P. 2000. Thinking objectively: software engineering in the small. *Communications of the ACM,* 43**,** 115-118.

FEREDAY, J. & MUIR-COCHRANE, E. 2008. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International journal of qualitative methods,* 5**,** 80-92.

FERNANDEZ, D. J. & FERNANDEZ, J. D. 2008. Agile Project Management - Agilism versus Traditional Approaches. *Journal of Computer Information Systems,* 49**,** 10-17.

FITZGERALD, B., HARTNETT, G. & CONBOY, K. 2006. Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems,* 15**,** 200-213.

FLAVELL, J. H. 1979. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist,* 34**,** 906.

FLYVBJERG, B. 2006. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry,* 12**,** 219-245.

FORSYTHE, D. E. 1999. "It's Just a Matter of Common Sense": Ethnography as Invisible Work. *Computer Supported Cooperative Work,* 1-2**,** 127-145.

FOWLER, F. J. 2009. *Survey Research Methods,* Los Angeles, USA, Sage.

GIDDENS, A. 1984. *The Constitution of Society,* Los Angeles, University of California Press.

GLAZER, H., DALTON, J., ANDERSON, D., KONRAD, M. D. & SHRUM, S. 2008. Cmmi or agile: Why not embrace both! *Software Engineering Process Management* [Online]. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1291&context=sei&sei-redir=1&referer=http%3A%2F%2Fscholar.google.co.za%2Fscholar%3Fq%3DCmmi%2Bor%2Bagile%253A%2BWhy%2Bnot%2Bembrace%2Bboth%2521%26btnG%3D%26hl%3Den%26as_sdt%3D0%252C5%26as_vis%3D1#search=%22Cmmi%20or%20agile%3A%20Why%20not%20embrace%20both%21%22, Viewed 13 August 2013.

GRANT, K., HACKNEY, R. & EDGAR, D. 2010. *Strategic Information Systems Management,* Hampshire, Cengage.

GREENING, D. R. 2010. Scaling Scrum to the Executive Level. *2010 Hawaii International Conference on System Science.* Hawaii.

GREGOR, S. 2006. The Nature of Theory in Information Systems. *MIS Quarterly,* 30**,** 611-642.

HABRA, N., ALEXANDRE, S., DESHARNAIS, J., LAPORTE, C. & RENAULT, A. 2008. Initiating software process improvement in very small enterprises: Experience with a light assessment tool. *Information and Software Technology,* 50**,** 763-771.

HANFORD, M. F. 2004. *Program Management: Different from project management* [Online]. IBM. Available: http://atabkamprofessionalservices.vpweb.ca/upload/Program%20management_%20Different%20from%20project%20management.pdf. Viewed 5 January 2013.

HIGGINS, D. & MIRZA, M. 2012. Considering Practice: a contemporary theoretical position towards social learning in the Small Firm. *The Irish Journal of Management,* 31**,** 1-18.

HIGHSMITH, J. A. 2000. Retiring Life Cycle Dinosaurs. Using Adaptive Software Development to meet the challenges of a high-speed, high-change environment. *Software Testing and Quality Engineering* [Online]. Viewed

HIGHSMITH, J. A. 2000a. *Adaptive Software Development. A Collaborative Approach to Managing Complex Systems,* New York, Dorset House Publishing.

HIGHSMITH, J. A. 2002a. Extreme Programming. *Agile Project Management Advisory Service* [Online]. Viewed

HIGHSMITH, J. A. 2002b. What is Agile Software Development? *Crosstalk. The Journal of Defense Software Engineering***,** 4-9.

HIGHSMITH, J. A. 2004. *Agile Project Management: Creating Innovative Projects,* Boston, USA, Pearson.

HINOJO, L. F. J. 2014. Agile, CMMI®, RUP®, ISO/IEC 12207...: is there a method in this madness? *ACM SIGSOFT Software Engineering Notes,* 39**,** 1-5.

HODA, R., NOBLE, J. & MARSHALL, S. Agile Project Management. NZCSRSC 2008, 2008 Christchurch, New Zealand.
HODA, R., NOBLE, J. & MARSHALL, S. Organizing self-organizing teams. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 1-8 May 2010 2010 Cape Town, South Africa. ACM, 285-294.
HODA, R., NOBLE, J. & MARSHALL, S. 2013. Self-Organizing Roles on Agile Software Development Teams. *IEEE Transactions on Software Engineering,* 39**,** 422-444.
HODGSON, D. E. 2004. Project Work: The Legacy of Bureaucratic Control in the Post-Bureaucratic Organization. *Organization,* 11**,** 81-100.
HODGSON, D. E. & CICMIL, S. 2008. The other side of projects: the case for critical project studies. *International Journal of Managing Projects in Business,* 1**,** 142-152.
HOEBEKE, L. 2000. Making Work Systems Better: A Practitioner's Reflections. *Belgium Sprouts: Working Papers on Information Systems,* 1**,** 1-128.
HOFFMAN, K., PAREJO, M., BESSANT, J. & PERREN, L. 1998. Small firms, R&D, technology and innovation in the UK: a literature review. *Technovation,* 18**,** 39-55.
HURTADO, J. A., BASTARRICA, M. C., OCHOA, S. F. & SIMMONDS, J. 2013. MDE software process lines in small companies. *Journal of Systems and Software,* 86**,** 1153-1171.
IGIRA, F. T. 2012. Information Systems Deployment as an Activity System. *In:* DWIVEDI, Y. K., WADE, M. B. & SCHENBERGER, S. I. (eds.) *Information Systems Theory. Explaining and Predicting Our Digital Society, Vol. 2.* New York, USA: Springer.
IGIRA, F. T. & GREGORY, J. 2009. Cultural historical activity theory. *In:* DWIVEDI, Y. K., LAI, B., WILLIAMS, M. D., SCHENBERGER, S. I. & WADE, M. B. (eds.) *Handbook of research on contemporary theoretical models in information systems.* Pennsylvania, USA: IGI Global.
IRVINE, H. & DEO, H. 2006. The Power of the lens: a comparative analysis of two views of the Fiji Development Bank. *Accounting, Auditing and Accountability Journal,* 19**,** 205-227.
JACKSON, M. C. 2006. Beyond problem structuring methods: reinventing the future of OR/MS. *Journal of Operational Research Society,* 57**,** 868-878.
JACKSON, M. C., MANSELL, G. J., FLOOD, R., BLACKHAM, R. B. & PROBERT, S. V. E. 1991. *Systems Thinking in Europe,* New York, Plenum Publishing Corporation.
JARZABKOWSKI, P. 2003. Strategic practices: an activity theory perspective on continuity and change. *Journal of Management studies,* 40**,** 23-55.
JAYAWARDENA, D. S. & EKANAYAKE, L. L. 2010. Adaptation Analysis of Agile Project Management for managing IT projects in Sri Lanka. *2010 International Conference on Advances in ICT for Emerging Regions (ICTer).* Colombo, Sri Lanka.
JOHNS, J. L. 1996. A concept analysis of trust. *Journal of Advanced Nursing,* 24**,** 76-83.
JONAS, D. 2010. Empowering project portfolio managers: How management involvement impacts project portfolio management performance. *International Journal of Project Management,* 28**,** 818-831.
JONSSON, H. 2012. Lean Software Development: A Systematic Review.
KAPLAN, A. 1998. *The Conduct of Inquiry. Methodology for Behavioral Science,* New Jersey, USA, Chandler Publishing.
KAPTELININ, V. & NARDI, B. (eds.) 2012. *Activity theory in HCI: Fundamentals and Reflections*: Morgan and Claypool.
KAPTELININ, V. & NARDI, B. A. Activity theory: basic concepts and applications. CHI'97 extended abstracts on Human factors in computing systems: looking to the future, 1997. ACM, 158-159.
KELLY, A. 2013. *The Agile Pyramid and the Learning View* [Online]. Available: http://www.allankelly.net/presentations/index.html. Viewed 07 May 2013.
KERZNER, H. 2013. *Project Management. A Systems Approach to Planning, Scheduling and Controlling.,* New Jersey, USA, John Wiley and Sons.

KIM, Y. S. Analyzing Scrum Agile Software Development with Development Process, Social Factor and Project Management Lenses. 13th Americas Conference on Information Systems, 10-12 August 2007 2007 Keystone, USA.

KINCHELOE, J. & MCLAREN, P. 2003. Rethinking Critical Theory and Qualitative Research. *In:* DENZIN, N. K. & LINCOLN, Y. S. (eds.) *The Landscape of Qualitative Research. Theories and Issues.* 2nd ed. California: Sage.

KLEIN, H. K. & MYERS, M. D. 1999. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly,* 23**,** 67-94.

KORPELA, M., MURSU, A. & SORIYAN, H. A. 2002. Information systems development as an activity. *Computer Supported Cooperative Work,* 11**,** 111-128.

KRISHNAN, V. & ULRICH, K. T. 2001. Product development decisions: A review of the literature. *Management science,* 47**,** 1-21.

KTATA, O. & LÉVESQUE, G. 2009a. Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects. *C3S2E-09.* Montreal, Canada: ACM.

KTATA, O. & LÉVESQUE, G. 2009b. Agile development: issues and avenues requiring a substantial enhancement of the business perspective in large projects. *2nd Canadian conference on computer science and software engineering.* Montreal, Canada: ACM.

KUUTTI, K. 1995. Activity theory as a potential framework for human-computer interaction research. *In:* NARDI, B. (ed.) *Context and consciousness: Activity theory and human-computer interaction.* New York, USA: MIT Press.

LAPORTE, C. Y., ALEXANDRE, S. & O'CONNOR, R. V. A Software Engineering Lifecycle Standard for Very Small Enterprises. Proceedings of 15Th European Conference on Software Process Improvement, September 3-5 2008 2008a Dublin, Ireland. Springer, 129.

LAPORTE, C. Y., ALEXANDRE, S. & RENAULT, A. 2008b. Developing international standards for very small enterprises. *Computer,* 41**,** 98-101.

LATOUR, B. 1987. *Science in Action,* Cambridge, Harvard University Press.

LEE, A. S. 1989. A Scientific Methodology for MIS Case Studies. *MIS Quarterly,* 13**,** 33-52.

LEE, A. S. 1991. Integrating Positivist and Interpretive Approaches to Organizational Research. *Organizational Science,* 2**,** 342-365.

LEE, A. S. & BASKERVILLE, R. L. 2003. Generalizing Generalizability in Information Systems Research. *Information Systems Research,* 14**,** 221-243.

LEE, G. & XIA, W. 2010. Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. *MIS Quarterly,* 34**,** 87-114.

LEE, S. & YONG, H. 2013. Agile Software Development Framework in a Small Project Environment. *Journal of Information Process System,* 9**,** 69-88.

LEFFINGWELL, L. L. C. 2013. *Scaled Agile Framework* [Online]. Available: http://scaledagileframework.com/. Viewed 13 August 2013.

LEI. 2009. *Lean Enterprise Institute* [Online]. Available: http://www.lean.org/. Viewed 25 May 2012.

LEONT'EV, A. N. 1981. *Problems of the development of the mind,* Moscow, USSR, Progress.

LEVIN, G. & GREEN, A. R. 2010. *Implementing Program Management,* Florida, USA, CRC Press.

LEYBOURNE, S. A. 2009. Improvisation and agile project management: a comparative consideration. *International Journal of Managing Projects in Business,* 2**,** 519-535.

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.

LLEWELYN, S. 2003. Methodological Issues: What Counts as Theory in qualitative management and accounting research? - Introducing five levels of theorizing. *Accounting, Auditing and Accountability,* 16**,** 662-708.

LUUKKONEN, I. & MYKKÄNEN, J. Analyzing Process Modelling as Work Activity. *In:* KELLER, C. & WIBERG, M., eds. Information Systems Research Seminar in Scandinavia, August 17-20 2012 Sigtuna, Swenden. Tapir Akademisk Forlag, 9-24.

LYCETT, M., RASSAU, A. & DANSON, J. 2004. Programme management: a critical review. *International Journal of Project Management,* 22**,** 289-299.

LYYTINEN, K. J. & NGWENYAMA, O. K. 1992. What does computer support for cooperative work mean? A structurational analysis of computer supported cooperative work. *Accounting, Management and Information Technologies,* 2**,** 19-37.

MAHNIC, V. & DRNOVSCEK, S. Agile Software Project Management with Scrum. EUNIS 2005 2005 Manchester, United Kingdom. Citeseer.

MARCAL, A. S. C., DE FREITAS, B. C. C., FURTADO SOARES, F. S. & BELCHIOR, A. D. Mapping cmmi project management process areas to scrum practices. Software Engineering Workshop, 2007. SEW 2007. 31st IEEE, 2007. IEEE, 13-22.

MARTIN, R. 2005. Validity Vs Reliability: Implications for Management. *Rotman Magazine.* Toronto, Canada: Rothman School of Management -Toronto University.

MARTIN, R. 2011. The Innovation Catalysts. *Harvard Business Review,* 89**,** 82-86.

MARTIN, R. & AUSTEN, H. 1999. The Art of Integrated Thinking. *Rothman Magazine.* Toronto, Canada: Rothman School of Management - University of Toronto.

MATOS, S. & LOPES, E. 2013. Prince2 or PMBOK–a question of choice.

MAYLOR, H., BRADY, T., COOKE-DAVIES, T. & HODGSON, D. 2006. From projectification to programmification. *International Journal of Project Management,* 24**,** 663-674.

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**,** 372-383.

MCNELY, B. J., GESTWICKI, P., BURKE, A. & GELMS, B. Articulating everyday actions: an activity theoretical approach to scrum. 30th ACM international conference on Design of communication, 3-5 October 2012 2012 Seattle, USA. ACM, 95-104.

MEREDITH, J. R. & MANTEL, S. J. 2011. *Project management: a managerial approach,* New Jersey, USA, John Wiley and Sons.

MERRIAM, S. B. 2002. *Introduction to Qualitative Research,* San Francisco, John Wiley and Sons.

MICTSETA 2013. Sector Skills Plan 2014-2019. *In:* THE MEDIA, I. A. C. T. S. E. A. T. A. (ed.). MICTSeta.

MIDDLETON, P. & JOYCE, D. 2010. Lean software management: BBC Worldwide case study. *IEEE Transactions on Engineering Management,* 59**,** 20-32.

MIETTINEN, O., MAZHELIS, O. & LUOMA, E. 2010. Managerial Growth Challenges in Small Software Firms: A Multiple-Case Study of Growth-Oriented Enterprises. *Software Business.* Springer.

MINGERS, J. 2004. Real-izing information systems: critical realism as an underpinning philosophy for information systems. *Information and Organization,* 14**,** 87-103.

MINGERS, J. 2008. Pluralism, Realism and Truth: The Keys to Knowledge in Information Systems Research. *international Journal of Information Technologies and the Systems Approach,* 1**,** 79-90.

MINGERS, J. 2012. Abduction: the missing link between deduction and induction. A comment on Ormerod's 'rational inference: deductive, inductive and probabilistic thinking'. *Journal of the Operational Research Society,* 63**,** 860-861.

MINGERS, J. & BROKLESBY, J. 1997. Multimethodology: for Mixing Towards a Framework Methodologies. *International Journal of Management Science,* 25**,** 489-509.

MINGERS, J. & ROSENHEAD, J. 2004. Problem structuring methods in action. *European Journal of Operational Research,* 152**,** 530-554.

MINTZBERG, H. 1979. An emerging strategy of" direct" research. *Administrative science quarterly,* 24**,** 582-589.

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**,** 1869-1890.

MNKANDLA, E. 2010. Agile Software Engineering. *In:* RAMACHANDRAN, M. & DECARVALHO, R. (eds.) *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Collaboration.* IGI Global.

MOE, N. B., DINGSOYR, T. & DYBA, T. 2010. A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology,* 52**,** 480-491.

MUJINGA, M. Privacy and Legal Issues in Cloud Computing-The SMME Position in South Africa. 11th Australian Information Security Management Conference, 2nd-4th December 2013 2013 Edith Cowan University, Perth, Western Australia. Research Online, 49-59.

MÜLLER, R., MARTINSUO, M. & BLOMQUIST, T. 2008. Project portfolio control and portfolio management performance in different contexts. *Project Management Journal,* 39**,** 28-42.

MWANZA, D. Where theory meets practice: A case for an Activity Theory based methodology to guide computer system design. *In:* MICHITAKA, H., ed. INTERACT'2001: Eighth IFIP TC 13 International Conference on Human–Computer Interaction, July 8-12 2001 Waseda University Conference Centre, Shinjuku, Toyko, Japan. IOS Press, Oxford, 342–349.

MWANZA, D. 2002. Conceptualising work activity for CAL systems design. *Journal of Computer Assisted Learning,* 18**,** 84-92.

MYERS, M. D. 1997. *Qualitative Research in Information Systems* [Online]. Association for Information Systems. Available: http://www.qual.auckland.ac.nz/. Viewed 06 July 2012.

MYERS, M. D. 2013. *Qualitative research in business and management,* London, UK, Sage.

MYERS, M. D. & NEWMAN, M. 2007. The Qualitative Interview in IS Research: Examining the Craft. *information and Organization,* 17**,** 2-26.

NARDI, B. A. 1996. Studying context: A comparison of activity theory, situated action models, and distributed cognition. *Context and consciousness: Activity theory and human-computer interaction.*

NERUR, S. & BALIJEPALLY, V. 2007. Theoretical reflections on agile development methodologies. *Communications of the ACM,* 50**,** 79-83.

NERUR, S., MAHAPATRA, R. & MANGALARAJ, G. 2005. Challenges of migrating to agile methodologies. *Communications of the ACM,* 48**,** 72-78.

NICOLAIDES, A. 2011. Entrepreneurship-the role of Higher Education in South Africa. *Educational Research,* 2**,** 2141-5161.

NORDIN, N., DEROS, B. M., WAHAB, D. A. & RAHMAN, M. N. A. 2012. A framework for organisational change management in lean manufacturing implementation. *International Journal of Services and Operations Management,* 12**,** 101-117.

NUOPPONEN, A. 2010. Methods of concept analysis-towards systematic concept analysis *LSP Journal-Language for special purposes, professional communication, knowledge management and cognition,* 1**,** 4-12.

OGC 2007. *Managing Successful Programmes,* London, UK, The Stationary Office.

OLAWALE, O. & GARWE, D. 2010. Obstacles to the growth of new SMEs in South Africa: A principal component analysis approach *African Journal of Business Management,* 4**,** 729-738.

OLIVEIRA, M., BITENCOURT, C., TEIXEIRA, E. & SANTOS, A. C. Thematic Content Analysis: Is There a Difference Between the Support Provided by the MAXQDA® and NVivo® Software Packages? Proceedings of the 12th European Conference on Research Methods for Business and Management Studies, 2013 Guimaraes, Portugal. Academic Conferences Limited, 304-314.

OLIVER, M. 1992. Changing the Social Relations of Research Production. *Disability, Handicap and Society,* 7**,** 101-114.

OLUGBARA, O. O. & NDHLOVU, B. N. 2014. Constructing Frugal Sales System for Small Enterprises. *The African Journal of Information Systems,* 6**,** 120-139.

ORMEROD, R. J. 2009. Rational inference: deductive, inductive and probabilistic thinking. *Journal of the Operational Research Society,* 61**,** 1207-1223.

ORMEROD, R. J. 2012. On abduction: a response to Mingers. *Journal of the Operational Research Society,* 63**,** 696-697.

P2M 2008. *A Guidebook of Project and Program Management for Enterprise Innovation,* Toyko, Japan, Project Management Professionals Certification Centre (PMCC).

PALVIA, P., MIDHA, V. & PINJANI, P. 2006. Research Models in Information Systems. *Communications of the Association for Information Systems,* 17**,** 1042-1063.

PANIZZOLO, R., GARENGO, P., SHARMA, M. K. & GORE, A. 2012. Lean manufacturing in developing countries: evidence from Indian SMEs. *Production Planning & Control,* 23**,** 769-788.

PAROLIA, N., JIANG, J. J., KLEIN, G. & SHEU, T. S. 2011. The contribution of resource interdependence to IT program performance: A social interdependence perspective. *International Journal of Project Management,* 29**,** 313-324.

PATTON, M. Q. 1990. *Qualitative Evaluation and Research Methods,* Carlifornia, Sage.

PELLEGRINELLI, S. 2011. What's in a name: Project or programme? *International Journal of Project Management,* 29**,** 232-240.

PELLEGRINELLI, S., PARTINGTON, D., HEMINGWAY, C., MOHDZAIN, Z. & SHAH, M. 2007. The importance of context in programme management: An empirical review of programme practices. *International Journal of Project Management,* 25**,** 41-55.

PETERSEN, K. & WOHLIN, C. 2009. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software,* 82**,** 1479-1490.

PFLEEGER, S. L. & ATLEE, J. M. 2010. *Software Engineering,* New Jersey, Pearson.

PHAM, A. & PHAM, P. 2012. *Scrum in Action,* Boston, USA, Course Technology-Cengage Learning.

PIKKARAINEN, M., HAIKARA, J., SALO, O., ABRAHAMSSON, P. & STILL, J. 2008. The impact of agile practices on communication in software development. *Empirical Software Engineering,* 13**,** 303-337.

PIKKARAINEN, M., SALO, O., KUUSELA, R. & ABRAHAMSSON, P. 2012. Strengths and barriers behind the successful agile deployment—insights from the three software intensive companies in Finland. *Empirical Software Engineering,* 17**,** 675-702.

PINO, F. J., PARDO, C., GARCÍA, F. & PIATTINI, M. 2010a. Assessment methodology for software process improvement in small organizations. *Information and Software Technology,* 52**,** 1044-1061.

PINO, F. J., PEDREIRA, O., GARCÍA, F., LUACES, M. R. & PIATTINI, M. 2010b. Using Scrum to guide the execution of software process improvement in small organizations. *Journal of Systems and Software,* 83**,** 1662-1677.

PMG 2012. Small Business Review Report. *In:* INDUSTRY, D. O. T. A. (ed.).

PMI 2008. *A Guide to the Project Mangement Body of Knowledge,* Newton Square, PMI.

PMI. 2013. *Organizational Project Management Maturity Model* [Online]. Available: https://www.pmi.org/en/Business-Solutions/Organizational-Project-Management.aspx. Viewed 14 April 2014.

PMI. 2014. *Project Management Institute* [Online]. Available: http://www.pmi.org/. Viewed 10 September 2014.

POPPENDIECK, M. & POPPENDIECK, T. 2009. *Leading Lean Software Development: Results are Not the Point,* Upper Saddle River, New Jersey, USA, Addison-Wesley.

PRESSMAN, R. 2010. *Software Engineering in Practice,* New York, McGraw Hill.

QSRINTERNATIONAL. 2013. *Software for Qualitative Research* [Online]. Available: http://www.qsrinternational.com/. Viewed 05 May 2013.

QUMER, A. & HENDERSON-SELLERS, B. 2008. A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software,* 81**,** 1899–1919.

RAMASESH, R. V. & BROWNING, T. R. 2014. A conceptual framework for tackling knowable unknown unknowns in project management. *Journal of Operations Management,* 32**,** 190-204.

REICH, B. H., SAUER, C. & GERONIMO, A. Business Value in IT Projects: A Theoretical Model. International Research Workshop on IT Project Management, December 12-13 2008 Paris, France. 96-114.
REMENYI, D. 2012. *Case Study Research,* London, Academic Publishing International.
RICHARDSON, I. & VON WANGENHEIM, G. C. 2007. Guest Editors' Introduction: Why are Small Software Organizations Different? *Software, IEEE,* 24**,** 18-22.
RISING, L. & JANOFF, N. S. 2000. The Scrum software development process for small teams. *Software, IEEE,* 17**,** 26-32.
RODGERS, B. L. 1989. Concepts, analysis and the development of nursing knowledge: the evolutionary cycle. *Journal of Advanced Nursing,* 14**,** 330-335.
RODGERS, B. L. 2000. Concept Analysis: An Evolutionary View. *In:* RODGERS, B. L. & KNAFL, K. A. (eds.) *Concept Development in Nursing: Foundations, Techniques and Applications.* Philadelphia, USA: W. B. Saunders.
ROWLANDS, B. H. 2005. Grounded in Practice: Using Interpretive Research to Build Theory. *The Electronic Jornal of Business Research Methodology,* 3**,** 81-92.
ROYCE, W. W. Managing the development of large software systems. proceedings of IEEE WESCON, 1970. Los Angeles, 1-9.
RULE, P. & VAUGHN, J. 2011. *Your Guide to Case Study Research,* Pretoria, RSA, van Schaik Publishers.
RUNESON, P., HOST, M., RAINER, A. & REGNELL, B. 2012. *Case study research in software engineering: Guidelines and examples,* New Jersey, USA, Wiley.
RUSSELL, A. C. 2012. Moral distress in neuroscience nursing: An evolutionary concept analysis. *Journal of Neuroscience Nursing,* 44**,** 15-24.
RYCROFT, R. W. & KASH, D. E. 2004. Self-organizing innovation networks: implications for globalization. *Technovation,* 24**,** 187-197.
SAASTAMOINEN, I. & TUKIAINEN, M. Software process improvement in small and medium sized software enterprises in eastern Finland: A state-of-the-practice study. 11th European Conference on Software Process Improvement, 2004 Trondheim, Norway. 69-78.
SAUER, C. & REICH, B. H. 2009. Rethinking IT Project Management: Evidence of a new mindset and its implications. *international Journal of Project Management* 27**,** 182-193.
SAYER, A. 1992. *Method in Social Science: A Realist Approach,* London, UK, Taylor & Francis.
SCALEDAGILEACADEMY. 2013. *Scaled Agile Academy* [Online]. Available: http://www.scaledagileacademy.com/. Viewed 10 November 2012
SCHACH, R. S. 2011. *Object-Orientated and Classical Software Engineering,* Upper Saddle River, New Jersey, USA, McGraw-Hill.
SCHRAW, G. & MOSHMAN, D. 1995. Metacognitive theories. *Educational psychology review,* 7**,** 351-371.
SCHULTZE, U. 2000. A Confessional Account of an Ethnography About Knowledge Work. *MIS Quarterly,* 24**,** 3-41.
SCHWABER, K. 2011. *The Enterprise and Scrum*, Microsoft Press.
SCHWABER, K. & BEEDLE, M. 2002. *Agile software development with Scrum,* Upper Saddle River, New Jersey, Prentice Hall.
SCHWABER, K. & SUTHERLAND, J. 2011. *The Scrum Guide,* Scrum.org. Available: http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011. Viewed 10 July 2013.
SCHWALBE, K. 2013. *Information Technology Project Management,* Boston, USA, Cengage Learning.
SCRUM.ORG. 2013. *Scrum.org* [Online]. Available: http://scrum.org/. Viewed 01 January 2013.
SEI. 2013. *Software Engineering Institute: Carnegie Mellon* [Online]. Available: http://www.sei.cmu.edu/. Viewed 17 January 2012.
SEWCHURRAN, K. & BROWN, I. Toward an Approach to Generate Forward-looking Theories using Systems Concepts. *In:* CHIASSON, M., HENFRIDSSON, O., KARSTEN, H. & DEGROSS, J. I., eds.

Researching the Future in Information Systems: IFIP WG 8.2 Working Conference, 2011 Turku, Finland. Springer.
SEWCHURRAN, K., DE LA HARPE, A., MWALEMBA, G. & MCKINNELL, J. 2012. Making sense of the IT/IS industry to enable better business enablement and competitiveness: A case-study of the Western Cape in South Africa. *4th Annual IEEE colloquium on Software Engineering.* Cape Town, South Africa.
SILVERMAN, D. 2010. *Doing Qualitative Research,* California, Sage.
SINGH, A. & SEWCHURRAN, K. Towards Improved Understanding of Learning Practices used in Scrum Software Development Projects. Joint International Conference on Engineering Education and Research and International Conference Information Technology, 8-12 December 2013 2013 Cape Town, South Africa. Cape Peninsula University of Cape Town, 173-183.
SINGH, A. & SEWCHURRAN, K. An Activity Theory View of Management Practices within a Scrum Software Development Environment. 5th International Conference on Education and Information Management, 21-22 June 2014 2014 Durban, South Africa. International Foundation for Research and Development, 161-168.
SKYTTNER, L. 1996. General systems theory: origin and hallmarks. *Kybernetes,* 25**,** 16-22.
SLIGER, M. & BRODERICK, S. 2008. *The Software Project Manager's Bridge to Agility,* Boston, Pearson.
SMITH, M. L. 2006. Overcoming theory-practice inconsistencies: Critical realism and information systems research. *Information and Organization,* 16**,** 191-211.
SOMMERVILLE, I. 2011. *Software Engineering,* Boston, Pearson.
STAHL, B. C. 2013. Interpretive accounts and fairy tales: a critical polemic against the empiricist bias in interpretive IS research. *European Journal of Information Systems***,** 1-11.
STAKE, R. E. 1995. *The Art of Case Study Research,* California, USA, Sage Pulications.
SUTHERLAND, J. Future of scrum: Parallel pipelining of sprints in complex projects. Agile Conference, 2005. , 2005 Denver, Colorado, USA. IEEE, 90-99.
SUTHERLAND, J. 2010. *Scrum Handbook,* Boston, USA, Scrum Institute Training Press.
SUTHERLAND, J., RUSENG JAKOBSEN, C. & JOHNSON, K. Scrum and cmmi level 5: The magic potion for code warriors. Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, 2008. IEEE, 466-466.
TAYLOR, P. S., GREER, D., COLEMAN, G., MCDAID, K. & KEENAN, F. 2008. Preparing small software companies for tailored agile method adoption: Minimally intrusive risk assessment. *Software Process: Improvement and Practice,* 13**,** 421-437.
THIRY, M. 2002. Combining value and project management into an effective programme management model. *International Journal of Project Management,* 20**,** 221-227.
THIRY, M. 2004. "For DAD": a programme management life-cycle process. *International Journal of Project Management,* 22**,** 245-252.
THIRY, M. 2010. *Program Management,* Surrey, UK, Ashgate Publishing Group.
THIRY, M. & DEGUIRE, M. 2007. Recent developments in project-based organisations. *International journal of project management,* 25**,** 649-658.
TILSON, D. & LYYTINEN, K. 2005. Making Broadband Wireless Services: An Actor-Network Study of the US Wireless Industry Standard Adoption. *Sprouts: Working Papers on Information Systems,* 5**,** 137-154.
TJALE, A. A. & BRUCE, J. 2007. A concept analysis of holistic nursing care in paediatric nursing. *Curationis,* 30**,** 45-52.
TOBIN, R. 2010. Descriptive Case Study. *In:* MILLS, A. J., DUREPOS, G. & WIEBE, E. (eds.) *Encyclopedia of Case Study Research.* California, USA: SAGE Publications.
TOOR, S. R. & OGUNLANA, S. O. 2010. Beyond the 'iron triangle': stakeholder perception of key performance indicators (KPIs) for large-scale public sector development projects. *International Journal of Project Management,* 28**,** 228-236.

UNGER, B. N., GEMÜNDEN, H. G. & AUBRY, M. 2012. The three roles of a project portfolio management office: their impact on portfolio management execution and success. *International Journal of Project Management,* 30**,** 608-620.

USMAN, M., SOOMRO, T. R. & BROHI, M. N. 2014. Embedding Project Management into XP, Scrum and RUP. *European Scientific Journal,* 10**,** 293-307.

VÄHÄNIITTY, J. & RAUTIAINEN, K. 2008a. Towards a Conceptual Framework and Tool Support for Linking Long-term Product and Business Planning with Agile Software Development. *30th International Conference on Software Engineering.* Leipzig, Germany.

VÄHÄNIITTY, J. & RAUTIAINEN, K. T. Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. Proceedings of the 1st international workshop on Software development governance, May 12 2008b Liepzig, Germany. ACM, 25-28.

VAISMORADI, M., TURUNEN, H. & BONDAS, T. 2013. Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing and Health Sciences,* 15**,** 398-405.

VAN VLIET, H. 2008. *Software Engineering,* Hoboken, Wiley.

VAN WAARDENBURG, G. & VAN VLIET, H. 2013. When agile meets the enterprise. *Information and Software Technology,* 55**,** 2154-2171.

VERSIONONE 2010. State of Agile Survey 2010.

VISAGIE, J. C. 1997. SMMEs' challenges in reconstructing South Africa. *Management Decision,* 35**,** 660-667.

VOSS, C., TSIKRIKTSIS, N. & FROHLICH, M. 2002. Case research in operations management. *international Journal of Operations and Production Management,* 22**,** 195-219.

VYGOTSKY, L. L. S. 1978. *Mind in society: The development of higher psychological processes,* New York, USA, Harvard University Press.

WADOOD, F. & SHAMSUDDIN, A. 2012. Innovation in VSMEs of Pakistan: What Next! *OIDA International Journal of Sustainable Development,* 3**,** 81-88.

WALKER, K. & AVANT, K. 1988. *Strategies for Theory Construction in Nursing,* London, UK, Appleton and Lange.

WALSHAM, G. 1995. Interpretive case studies in IS research: nature and method. *European Journal of information systems,* 4**,** 74-81.

WALSHAM, G. 2006a. Doing interpretive research. *European journal of information systems,* 15**,** 320-330.

WALSHAM, G. 2006b. Doing Interpretive Research. *European Journal of Information Systems,* 15**,** 320-330.

WANG, L. 2007. Agility counts in developing small-size software. *IEEE Potentials,* 26**,** 16-23.

WANG, X., CONBOY, K. & CAWLEY, O. 2012. "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software,* 85**,** 1287-1299.

WEICK, K. E. 1989. Theory Construction as Disciplined Imagination. *The Academy of Management Review,* 14**,** 516-531.

WELFORD, C., MURPHY, K., WALLACE, M. & CASEY, D. 2010. A concept analysis of autonomy for older people in residential care. *Journal of clinical nursing,* 19**,** 1226-1235.

WEST, D. & GRANT, T. 2010. Agile Development: Mainstream Adoption Has Changed Agility. Forrester.

WHETTEN, D. A. 1989. What Constitutes a Theoretical Contribution? *The Academy of Management Review,* 14**,** 490-495.

WHITAKER, K. 2010. *Principles of Software Development Leadership. Applying Project Management Principles to Agile Software Development.,* Boston, Course Technology: Cengage.

WILLIAMS, L. & COCKBURN, A. 2003. Agile software development: It's about feedback and change. *Computer,* 36**,** 39-43.

WILSON, J. 1963. *Thinking with concepts,* New York, USA, Cambridge University Press.
WINTER, M., SMITH, C., MORRIS, P. & CICMIL, S. 2006. Directions for future research in project management: The main findings of a UK government-funded research network. *international Journal of Project Management,* 24**,** 638-649.
WOMACK, J. P. & DANIEL, T. J. 2003. *Lean Thinking: Banish Waste and create wealth in your Corporation,* New York, USA, Free Press.
WUISMAN, J. J. J. M. 2005. The Logic of Scientific Discovery in Critical Realist Social Scientific Research. *Journal of Critical Realism,* 4**,** 366-394.
YANG, H., HUFF, S. & STRODE, D. 2009. Leadership in Software Development: Comparing Perceptions of Agile and Traditional Project Managers. *15th Americas Conference on Information Systems.* San Francisco, California.
YIN, R. K. 2003. *Case Study Research: Design and Methods,* California, Sage.
YIN, R. K. 2009. *Case Study Research Design and Methods,* California, USA, Sage Publications Inc.
YIN, R. K. 2011. *Qualitative Research from Start to Finish,* New York, USA, The Guilford Press.
YIN, R. K. 2012. *Applications of Case Study Research,* Calafornia, USA, Sage Publications.
YOUNG, M. & CONBOY, K. 2013. Contemporary project portfolio management: Reflections on the development of an Australian Competency Standard for Project Portfolio Management. *International Journal of Project Management,* 31**,** 1089-1100.
ZULKOSKY, K. 2009. Self-efficacy: a concept analysis. *Nursing Forum,* 44**,** 93-102.

Appendix

Table A1 depicts the roles and responsibilities most likely to be found in a Scrum software development environment.

Table A1: Typical Personnel Roles in Scrum

Scrum Role	Responsibility
Product Owner	A representative and voice for the client. This is a person that is either external to the development team, i.e. from another company or department; or internal, i.e. a business analyst who is part of the team and has extensive domain knowledge of client business. The Product Owner has knowledge of the functionality of the required software and its priority.
Scrum Master	Often described as a servant-leader, the Scrum Master strives to resolve challenges facing the development team. He/she also ensures the principles and practices of Scrum are adhered to during all stages of development.
Developers	Are responsible for delivering working software. Typical duties include user story development, sprint planning, designing, coding, testing, implementing, and documentation development.

As mentioned previously, Scrum is designed as a framework of best practice guidelines and is mostly devoid of strict rules. Therefore, practical application may result in additional roles or slightly altered responsibilities.

The major Scrum artifacts and metrics for tracking project progress are shown in Table A2.

Table A2: Typical artifacts and metrics of Scrum

Scrum Artifact/Metric	Purpose
User Story	User story is a functional or a business requirement that the software product needs to satisfy. During the early stages of the Scrum project a collection of user stories will be captured on the product backlog. The software team will develop the software product such that each user story is satisfied.
User Story Points	User stories have varying levels of complexity in development. User story point is a numerical or some other indication of the complexity.
User Story Priority	User stories have varying levels of importance. A priority is a numerical or some other indication of importance.
Burndown Chart	The Burndown chart provides an indication of overall progress and predicts time remaining for a sprint or the whole project. These values are derived from the completed user stories and the end date.

Velocity Chart/Velocity	The Velocity Chart tracks the user story points completed within a timeframe (usually for a sprint). It provides an indication of how well a team is keeping to schedule. Velocity is the measure of team performance over time.
Kanban Board	Kanban is a Japanese term which translates roughly to signal card. In the Scrum context, a Kanban Board is used to visualize work. This is done by showing user stories completed, those still in progress and those still awaiting activation. Kanban Board is a popular artifact that serves as a focal point for daily stand-up meetings (see Table 6).
Code Coverage	A value that represents the proportion of developed software code that has been tested.

Table A3 highlights typical practices and associated activities in Scrum.

Table A3: Typical practices performed in Scrum

Scrum Practice	***Activities***
Create Product Backlog	Product Owner, together with other stakeholders, decides on business requirements. This is communicated as a prioritized set of user stories on a product backlog. Product backlog evolves throughout the project to reflect the most recent understanding of client needs.
Sprint	A product development cycle that usually lasts 3-4 weeks. Start and end dates are fixed.
Sprint Planning	Occurs before each sprint. The development team members examine priority and then decide on the user stories from the product backlog to be developed during the sprint. The team may seek clarity on user story requirements from the Product Owner before selecting user stories. Each user story is decomposed into smaller units of work called tasks.
Daily Stand-up meeting (Scrum)	Short daily meeting where development team presents work done, work in progress and report on difficulty. Scrum Master is usually chair.
Sprint Review	Demonstration of software completed to date to client. Product Owner, developers, Scrum Master and clients and other stakeholders may be present. What should be developed next is discussed thereafter.
Sprint Retrospective	Development team reflects on the sprint and identifies areas in need of improvement. Interventions are planned and will be implemented in the next sprint.
Test-Driven Development (TDD)	A development practice where test cases are first devised followed by writing code. The test case specifies a measure of functional aspect. Code is written to satisfy or “pass” the test cases.