A SOFTWARE DEVELOPMENT METHODOLOGY FOR SOLO SOFTWARE DEVELOPERS: LEVERAGING THE PRODUCT QUALITY OF INDEPENDENT DEVELOPERS

BY

SIBONILE MOYO

submitted in accordance with the requirements for

the degree of

DOCTOR OF PHILOSOPHY

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR

PROFESSOR E. MNKANDLA

February 2020

DECLARATION

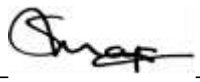
Name: _____Sibonile Moyo _____

Student number: _____61514780_____

Degree: _____Doctor of Philosophy in Computer Science_____

A Software Development Methodology for Solo Software Developers: Leveraging the Product Quality of Independent Developers

I declare that the above thesis is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

__  _____ __ February 2020_____

SIGNATURE DATE

ABSTRACT

Software security for agile methods, particularly for those designed for individual developers, is still a major concern. With most software products deployed over the Internet, security as a key component of software quality has become a major problem. In addressing this problem, this research proposes a solo software development methodology (SSDM) that uses as minimum resources as possible, at the same time conforming to the best practice for delivering secure and high-quality software products.

Agile methods have excelled on delivering timely and quality software. At the same time research also shows that most agile methods do not address the problem of security in the developed software. A metasynthesis of SSDMs conducted in this thesis confirmed the lack practices that promote security in the developed software product. On the other hand, some researchers have demonstrated the feasibility of incorporating existing lightweight security practices into agile methods.

This research uses Design Science Research (DSR) to build, demonstrate and evaluate a lightweight SSDM. Using an algorithm adapted for the purpose, the research systematically integrates lightweight security and quality practices to produce an agile secure-solo software development methodology (Secure-SSDM). A multiple-case study in an academic and industry setting is conducted to demonstrate and evaluate the utility of the methodology. This demonstration and evaluation thereof, indicates the applicability of the methodology in building high-quality and secure software products. Theoretical evaluation of the agility of the Secure-SSDM using the four-dimensional analytical tool (4-DAT) shows satisfactory compliance of the methodology with agile principles.

The main contributions in this thesis are: the Secure-SSDM, which entails description of the concepts, modelling languages, stages, tasks, tools and techniques; generation of a quality theory on practices that promote quality in a solo software development environment; adaptation of Keramati and Mirian-Hosseinabadi's algorithm for the purposes of integrating quality and security practices. This research would be of value to researchers as it introduces the security component of software quality into a solo software development environment, probing more research in the area. To software developers the research has provided a lightweight methodology that builds quality and security into the product using minimum resources.

Keywords: agile methods; agility degree; design science research; industry developers, software development methodology; software product quality; software quality; software security; solo software development.

ACKNOWLEDGEMENTS

First and foremost, I wish to thank my supervisor, Professor Ernest Mnkandla for his dedication in supervising this research. He has patiently guided and empowered me to do and complete my research. He promptly gave feedback and direction whenever there was need to and showed me the direction towards success. In the same vein, I wish to thank the UNISA School of Computing for various forms of support given to this research.

I also thank my family for giving me the time to concentrate on this research. In particular, I express my greatest gratitude to my husband Mahubo Moyo for taking care of the family while I dedicated my time to this research. I thank my lovely daughters Thubelihle, Nomakhosi and Nomqhele Moyo for looking after themselves while I concentrated on this work.

My greatest appreciation also goes the National University of Science and Technology (NUST), Zimbabwe, for sponsoring my PhD studies. Apart from providing the financial support needed to complete the study, the institution also funded my travel to conferences that were key to the success of this study. Further, I wish to thank the NUST gate keeper, the registrar for clearing me to conduct this research in the institution. Last but not least, I wish to thank all the participants who took part in the demonstration and evaluation part of this methodology. These include the Computer Science Part II class of 2018 from NUST, and the three expert developers from industry. These participants helped to shape the Secure-SSDM to be what it is in this thesis.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

4-DAT	4-Dimensional Analytical Tool
C	Compatible
ICS	Internal communication system
ICTS	Information communication services
NC	Not compatible
NUST	National University of Science and Technology
PXP1	Personal eXtreme Programming developed by Agarwal and Umphress
PXP2	Personal eXtreme Programming by Dzhurov, Krasteva, and Ilieva
OOSCD	Object oriented software concepts and development
QA	Quality assurance
SAD	Systems Analysis and Design
SDM	Software development methodology
Secure-SSDM	Secure solo software development methodology
SSD	Solo software development
SSDM	Solo software development methodology
VSD	Very small-scale development

PUBLICATIONS FROM THIS THESIS

Moyo, S. & Mnkandla, E. (2019). A Metasynthesis of Solo Software Development Methodologies. IEEE International Multidisciplinary Information Technology and Engineering Conference (IEEE-IMITEC 2019), Vanderbijlpark, South Africa.

Moyo, S. & Mnkandla, E. (2020) 'A Novel Lightweight Solo Software Development Methodology with Optimum Security Practices', *IEEE Access*, 8, pp. 33735–33747. doi: 10.1109/ACCESS.2020.297100.

CHAPTER 1 INTRODUCTION

1.1 Background

The main aim of software engineering is to develop methods so as to inform and improve practice (Dittrich 2016, p.221). Software development methodologies (SDMs) as part of software engineering research, seek to achieve this aim through improving the analysis, design, testing, implementation and maintenance of software. A high-quality SDM produces a high-quality software product (Sommerville 2011; Pressman & Maxim 2015).

Several definitions of SDMs exist. An SDM is a systematic approach to software development that incorporates system models, notations, rules, and design advice towards the production of high-quality software (Sommerville 2011). Terms such as method, software process model and software development process are at times used interchangeably with SDM. Defined as a method, an SDM is an explicit description of an approach to software development specifying stages, tasks, products, roles and actions associated with the development process (Dittrich 2016, p.226). Pressman and Maxim (2015, p.40) define a software process model as a set of activities and tasks, together with their organisation to deliver quality software. In a way, a software development methodology organises the software development process so that it produces high-quality software.

Two broad classes of SDMs exist. These are traditional and agile methods. Traditional methods emerged as a solution to the software crisis (Naur & Randell 1968). Designed to bring order into the software development process, these tend to be prescriptive, heavyweight and associated with a lot of documentation. The documentation guides and ensures that software developers systematically navigate the systems development life cycle (SDLC). Developers' activities are recorded in prescribed documents and in a particular format. Examples of such methodologies include the Waterfall model, the V-model and, the Spiral model just to name a few. Agile methods on the other hand are less prescriptive and lightweight. These have since gained popularity due to their ability to deal with the changing development environment, reduced development costs and reduced time to market (Nurdiani *et al.* 2019, p.1). Popular representatives of agile methods include eXtreme Programming (XP) (Beck & Andres 2004), Scrum (Schwaber 1997), and the Crystal family (Cockburn 2004).

SDMs are further classified as either personal or team-based. Personal SDMs support independent developers in their quest for producing quality software. Activities in these methods are organised such that the various roles in the development process are played by an individual working alone. They are designed to address the unique needs of a solo developer. A seminal example of Personal SDMs is the Personal Software Process (PSP) (Humphrey 1995). Team-based SDMs on the other hand are targeted at coordinating various roles in a software project. These define different roles and responsibilities in the team. Focus here is made on defining communication channels among team members and coordination of the various members towards the delivery of high-quality software. This thesis focuses on personal SDMs.

Most research on SDMs has focussed on team-based methods at the expense of methods designed for individuals (Agarwal & Umphress 2008; Bernabé, Navia & García-Peñalvo 2015, Dzhurov, Krasteva & Ilieva 2009; Kruchten 2002). These individuals, also known as solo developers or freelancers, have the sole responsibility of delivering quality software. The delivered product is usually small to medium size, and in some cases, a component of a larger product. Solo developers contribute remarkably to the design of software in the market today. Their contribution can be seen both in the open source community and commercial software. Section 1.3 elaborates on this contribution and shows why it is necessary for researchers to focus on this lot of developers as well.

1.2 Software Quality

Software quality is a core component of a successful software development project. Many definitions of software quality exist (Sfetsos & Stamelos 2010, p.44; García-Mireles et al. 2012, p.134). García-Mireles *et al.* (2015, p.150) define software quality from a software product and software process perspective. The software process perspective considers the capability of a process to deliver quality software. This perspective upholds that a high-quality process produces a high-quality software product. Methods, activities, tools and techniques are defined within the development process to support product quality (Fuggetta 2000).

The software product perspective considers software quality to be the expected quality characteristics of a product, derived from a particular quality model (García-Mireles et al. 2015, p.150). These characteristics form part of the non-functional requirements of the software product (Nistala et al. 2016, p. 134). The quality model in this case serves as a basis

for evaluating the product quality. A quality product is therefore expected to portray in addition to functional requirements, these non-functional requirements (Kadi et al. 2016).

According to Nistala et al. (2016 p.134), software quality is simply the ability of a software product to meet (both stated and implied) requirements. For those requirements, identified and agreed upon by project stakeholders, appropriate practices for developing the product are enacted and monitored to attain the required quality. In most cases, a separate quality assurance team is set to monitor the development team's adherence to the expectations. Separating the development team from the quality assurance team is a traditional approach to software quality assurance (Marchewka 2015 pp.242-246). Agile methods have a different approach to assuring software quality. Quality assurance (QA) techniques in agile methods are normally embedded in the software development process (Mnkandla & Dwolatzky 2007, pp.8-9; Sfetsos & Stamelos 2010, p.44; Janus et al. 2012, p.12). Embedding quality practices in the software development method transfers the responsibility of QA to the software development team (Janus et al. 2012, pp. 11-12). Agile methods empower development teams to both establish software requirements and to ensure that quality is built into the resulting software product. This team empowerment is most ideal for solo development environments where the developer has to play both the development and quality assurance roles.

This research adopts the agile approach to building quality into the designed software product. A generic agile SSDM that embeds quality and security practices and techniques to promote building of high-quality software products is proposed. The proposed Secure-SSDM is designed to be lightweight to address the unique characteristics of the solo development environment. The solo development environment is characterised by limited resources (human, financial and technical) (Basri & O'Connor 2010). Besides the limited resources, the solo development environment is also associated with fast development speed and multitasking as developers often have to work on several projects at the same time. Further, in a solo development environment, peer review, which is an important component of quality, is not readily available. Section 1.3 details the characteristics of the solo software development environment.

1.3 The Solo Software Development Environment

In a solo software project, one person takes on the full responsibility of the development process in the project. The success of the development effort is heavily dependent on the solo

developer. The developer is responsible for every technical aspect of the software project and the resulting product. Usually the developer assumes various roles during the software development process, which in most cases requires self-criticism to ensure quality in both the process and the resulting product. Solo developers have to work closely with the users as these are their only source of readily available peer review.

Solo software development (SSD) dates back to the 1960s. This is the code and fix era of individual (cowboy) programmers who could spend the whole night fixing errors in code (Boehm 2006, p.14). These cowboys' success at fixing the errors would then be celebrated by the rest of the team after development resumes during the day. While cowboys in that era were part of a team, the cowboy approach to software development has since evolved to freelance software development. Instead of being part of a team, most freelancers develop software as individuals.

Freelance (solo) software development is a growing industry, particularly in developing countries as it addresses the problems of unemployment and those of high transportation costs (Haq *et al.* 2018). The growth in freelance software development is seen in the upsurge of freelancers in the mobile applications industry (Hsieh & Hsieh 2013, p.309). Developers in this industry contribute a number of innovative solutions such as gaming applications, health management, and business management applications, among others. Further, the increase in the numbers of websites that advertise these is another indicator of the popularity of this industry (Ahmed & Hoven 2010, p.416). Global examples of websites advertising software development freelancers include Toptal, Upwork, Guru and Freelancer (Steiner 2015), just to name a few.

South Africa like all developing countries has also seen a remarkable growth in the freelance software industry. This is evident from the number of websites linking freelancers with prospective clients. South African websites engaged in freelance business include but are not limited to: Hire a programmer; Toptal South Africa; and Payperproject. Hire a programmer classifies developers into Web (450 profiles), App (180 profiles), Database (480 profiles) and Desktop (450 profiles) developers (*Hap* 2020). While these numbers of profiles are not necessary mutually exclusive, (as most desktop developers would also qualify as database developers) this website indicates a viable industry. Toptal South Africa, like its global counterpart emphasizes in providing top talent programmers to clients locally and globally. Most developers advertising their skills in this site indicate whether they are available

remotely or onsite. This is the favourable characteristic of freelancing as it means developers can be employed from anywhere.

As developers engage in a global market, it is important that these freelancers are equipped with skills that enhance their competitiveness in this market. Most freelance websites provide a rating facility that reflects customer satisfaction on the services provided by the developer. The freelancer's rating increases their chances of being hired. Solo developers adopting the necessary quality and secure software development skills improve the quality of their software products. This in turn improves customer satisfaction and developer rating. This thesis proposes a secure software development methodology, that can be adopted by freelance developers seeking to improve the quality and security of their software products, at the same time enabling them to gain a competitive advantage in the software development industry.

The solo development environment unlike the team environment has its unique characteristics that impact on the quality of the developed software products (Laporte et al. 2006, p.3; ISO/IEC 2014, p1) :-

i. Limited resources – where the developer is the sole owner of the development house, resources tend to be limited (Wongsai et al. 2015, p.14; Keshta & Morgan 2017, p.24163). The little resources are therefore solely used to support activities directly linked to the development of the product (Coleman & O'Connor 2008, p.773; Basri & O'Connor 2010, p.1457).

ii. Minimal knowledge management on the development process – Software development is heavily dependent on knowledge management. Knowledge from past projects inform decisions on current projects. Due to limited resources coupled with fast development speed, solo developers may not have the capacity and time to maintain a database of past projects (Paternoster et al. 2014, p.2).

iii. Fast development time – The current software development environment demands fast software product delivery. Apart from dealing with fast development speed that characterises today's software industry in general, solo developers need to deal with the execution of simultaneous projects for survival in the market (Bernabé, Navia & García-Peñalvo 2015, p. 687).

These unique features of this environment are the main reason why there is need to develop methods tailored for such an environment.

Several researchers have tackled the problem of developing methods for such an environment. Both heavyweight and lightweight methods have been designed for the purpose. As indicated in Section 1.1, PSP is a well-established process, designed to support independent developers working on individual sized software modules. Developers using PSP perform design and code reviews, with the aim of removing most of the defects before software testing. In doing so, developers record their data on identified defects on logs, which are then used to plan future development efforts (Humphrey 2000). Various studies have confirmed the utility of PSP in designing quality software products (Abrahamsson *et al.* 2002; Pressman & Maxim 2015).

While PSP's utility in developing quality software products has been empirically established, its main disadvantage is that it is document heavy. Due to its heavy documentation processes, its complexity, lengthy training sessions and high training costs, PSP has not been widely adopted in industry (Pressman & Maxim 2015). Further, PSP is designed to prepare developers to fit into a Team software process (TSP) environment, and not necessarily to continue in a solo environment. In response to the short comings of PSP, some researchers (Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009; Bernabé, Navia & García-Peñalvo 2015; González-Sanabria, Morente-Molinera & Castro-Romero 2017) have developed agile SSDMs. Developing agile SSDMs is a growing research interest as is evident from the cited publications. However, while this research area is attracting a number of researchers, research efforts on SSDMs have not fully addressed the problem of developing quality software. One of the quality aspects that have not been fully addressed by these methods is that of security.

Most agile methods lack features designed to build security into the software product (Ayalew, Kidane & Carlsson 2013; Firdaus, Ghani & Jeong 2014; Ghani, Azham & Jeong 2014; Othmane *et al.* 2014; Rafi *et al.* 2015). From the literature reviewed in this thesis, no research has tackled the problem of incorporating security practices into the development process in an SSDM context. With the increase in the adoption of agile methods in the software development practice, the lack of security in agile methods becomes a great concern. This is further fuelled by the increase in both the numbers and complexity of security threats to individual and organisational assets. With most services deployed over the Internet, security consideration becomes a must in the software development process.

This research utilises existing lightweight SSDMs to derive best practices in developing software in the solo development environment. The research posits that, using an appropriate methodology, the quality practices in the SSDM knowledge base can be synthesised to produce a higher quality SSDM (Peffers et al. 2008, p.49). Having shown using a metasynthesis conducted in Chapter 2 that existing SSDMs lack security practices, the research draws lightweight security practices from secure software development methods. The identified security practices are systematically integrated with the quality practices from the SSDMs to design the proposed Secure-SSDM.

1.4 Problem Statement

Software development methodology research has focused on large and small scale development at the expense of individual (solo) software development (Hollar 2006; Bernabé, Navia & García-Peñalvo 2015; Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009). Further, the few existing lightweight SSDMs do not address the security aspect of the developed software. This lack of security promoting practices in agile methods in general, is corroborated by a number of researchers (Beznosov & Kruchten 2004; Ghani, Azham & Jeong 2014; Baca *et al.* 2015; Aguda 2016). Insecure software development methodologies build insecure software products (Homaei & Shahriari 2019).

In trying to address the problem of insecure software development, this research proposes an agile Secure-SSDM, designed to improve the quality and security of software developed by solo developers. Using the DSR methodology, lightweight quality and security practices are identified from the SSDM and secure software development literature respectively. The identified practices are used to create a higher quality methodology with practices that promote quality and security in the developed software. Keramati and Mirian-Hosseinabadi's algorithm is adapted for the purposes of slyly integrating core quality practices with security practices while maintaining the agility of the resulting practices.

1.5 Research Aim

The aim of this research is to design and implement a secure-solo software development methodology (Secure-SSDM) that covers the complete SDLC. The methodology is designed through the identification and integration of quality with security promoting practices, tools and techniques from existing SSDMs and secure software processes respectively. A satisficing design of the proposed Secure-SSDM is produced to meet the solo developers' requirements.

The main contribution of this thesis is the Secure-SSDM which entails description of the concepts, modelling languages, stages, tasks, tools and techniques (Dittrich 2016). The designed methodology is unique for the solo environment in that it incorporates security promoting practices which are not present in the current SSDMs. The second contribution is the generation of the theory on how the methodology promotes quality in the developed software (Hevner *et al.* 2004). A third contribution is the adaptation of an existing algorithm to systematically integrate quality practices with lightweight security practices. In integrating the two types of practices, care is taken not to compromise the agility of the resulting methodology. These contributions are elaborated in Chapter 7.

1.6 Research Questions

In order to address the foregoing problem, the research provides answers to the following research question (RQ): -

RQ. How can a lightweight solo software development methodology be designed to use as minimum resources as possible, at the same time conforming to the best practice for delivering secure, high-quality software products?

A Secure-SSDM was developed through integrating quality practices extracted from existing SSDMs with lightweight security practices extracted from secure software development methodologies. A multiple-case study and the 4-DAT framework were used to evaluate the utility and agility of the methodology.

To answer the main question, the following sub-questions were pursued: -

SQ1. What methodologies exist for lightweight solo software development?

SQ2. What software development strategies and techniques in the identified methodologies promote quality in the developed software?

SQ3. What lightweight practices and techniques in the software development life cycle promote security in the developed software?

SQ4. How can quality and security practices from lightweight software development methodologies be synthesised into a solo software development methodology that promotes quality and security in the developed software?

SQ5. How can the resulting methodology be evaluated?

1.7 Research Objectives

Motivated by the questions raised in Section 1.6, the objectives of this study can be summarised as follows:

i. **To explore the existing lightweight solo development methodologies**.

A systematic literature review of lightweight solo software development methods was conducted. This facilitates the understanding of the current methodologies and their focus. The literature was used to fully expose the gap to be filled by this research. Processes, practices, techniques and tools for software development were explored. Approaches to methodology design and development were reviewed for the production of a high-quality methodology. The literature survey is discussed in Chapter 2.

ii. **To analyse existing methodologies' practices designed to enable quality in the developed software**

Using metasynthesis, quality practices from lightweight SSDMs were identified, analysed and organised into a framework for solo software development. The ISO/IEC 25010 quality model was used to assess the quality of the resulting framework. The framework has been iteratively refined to produce a desirable base for the formulation of the Secure-SSDM. The analysis is performed in Chapters 2 and 4.

iii. **To identify lightweight security practices from existing lightweight methodologies**

Secure software development literature was reviewed to identify security promoting practices for possible integration with quality practices in the framework from (ii). A systematic literature review by Rindell, Hyrynsalmi and Leppänen (2017) was used as a source to identify literature discussing secure software development together with associated security practices. The security practices are analysed in Chapter 4.

iv. **To synthesise the lightweight quality and security practices to produce secure quality software development practices**.

Using Keramati and Mirian-Hosseinabadi's adapted algorithm, quality and security practices were synthesised into secure-quality practices to produce the secure-software development methodology. A comprehensive description of the methodology was provided, together with guidelines on methodology application. Techniques, tools and deliverables from the

methodology stages were fully documented. The synthesis of the practices is performed in Chapter 5.

v. **To evaluate the utility of the resulting methodology through the development of software products in an industry setting.**

The Secure-SSDM was theoretically evaluated using the 4-DAT framework, and naturally evaluated using a multiple-case study. The theoretical evaluation focused on assessing the agility of the artefact while the natural evaluation focused on the utility of the method. The first case study was conducted in an academic setting, with the second one conducted in an industry setting with solo software developers in and around Bulawayo, Zimbabwe. Solo developers in these two settings were asked to use the methodology to develop software products. Qualitative data on the perceptions of solo developers using the methodology was collected and analysed qualitatively. Results from the study show that the Secure-SSDM can be used to develop high-quality and secure software products. The evaluation is performed in Chapter 6.

1.8 Research Methodology

A research methodology provides a systematic means for undertaking the research. A research methodology is premised on the research paradigm adopted for the research. In this thesis DSR was adopted as the overarching paradigm. DSR was used to identify the problem, propose and evaluate the solution for its utility. DSR was complemented by the Interpretivist paradigm for the purposes of dealing with the perceptions of the freelance developers at the conception and evaluations stages of the research.

Following closely the DSR methodology, the Secure-SSDM was designed incrementally and iteratively, with every iteration constituting methodology refinement. First, quality and security practices were separately drawn from the existing SSDMs and small-scale SDMs knowledge bases respectively. These were then integrated using an algorithm adopted and adapted for the purpose. The Secure-SSDM was then applied in an academic setting. A focus group discussion and document analysis were used to collect the perceptions of the student developers on the methodology. Data was analysed qualitatively. Feedback obtained from participants' views on the utility of the methodology in building quality and secure software was used to refine the Secure-SSDM.

The refined version of the Secure-SSDM was applied in an industry setting. Three developers used the methodology in developing software products of their choice. Interviews were then held with the developers to collect their perceptions on the methodology. Individual member checking of the collected data was conducted through electronic mail. This was done to ensure the reliability of the findings of the study. Feedback from the participants was analysed qualitatively. At the conclusion of the research, a feedback meeting was held with the three expert developers. This was done to minimise researcher bias and to improve the accuracy of the interpretation of the participants' perceptions (Santos, Magalhãe & da Silva 2017, p. 188). The Interpretivist approach guided the data collection and analysis during the evaluation of the primary and final versions of the Secure-SSDM. A theoretical evaluation was also performed to assess conformance of the methodology with agile principles. This provided for the rigour that is characteristic of DSR.

1.9 Ethical Considerations

The Secure-SSDM's utility and quality were evaluated both theoretically and empirically. For the empirical evaluation, the methodology was used by developers to design software products in a multiple-case study. Interviews and focus group discussions were conducted to obtain the developers' perceptions on the utility of the methodology. For the academic case study, clearance was sought with the university gate keeper before conducting the research. Further, an informed consent from each of the industry developers was obtained. Using the university gate keeper's letter and the informed consent letters from the developers, an ethical clearance with UNISA was obtained. The gate keeper letter of clearance and the UNISA ethical clearance are attached in appendix A.

1.10 Justification of the Research

As software continues to penetrate various aspects of human life, product quality becomes of paramount importance to both its users and business. High-quality and secure software has a positive impact on its users, and the business environment. Software developers are therefore indebted to deliver high-quality software to their users and business if this positive impact is to be achieved. SDMs enhance the quality of software products through incorporating practices for building quality and security into the resulting software product. In this research, such practices are referred to as, quality promoting practices.

The advent of mobile and web-based applications has led to an increase in the solo development environment. Due to their size, these applications can easily be handled by an individual working alone. At the same time research shows that a number of design flaws during web applications development contribute remarkably to security breaches in web applications (OWASP 2006, 2017; Hakim, Sellami & Abdallah 2016). Security breaches on websites result in loss of assets and has a negative impact on both individuals and business (Hakim, Sellami & Abdallah 2016, p.182). As the mobile and web applications industries continue to grow, so will the need for solo software development methods. The arguments raised in this paragraph point to the need of developing methods that can be used by individual developers in enhancing the quality and particularly the security of their software products.

Besides the mobile and web applications development environment, particular open source environments such as the Ruby on Rails community, thrive on contributions of software components (gems) from solo developers known as lone wolves. Gems are a key component of the Rails ecosystem as they are used as components in a number of software products. A lone wolf in the Rails ecosystem is a solitary developer that has produced the most important gems for the ecosystem, independent of other developers. An analysis of the Ruby software development ecosystem by Kabbedijk and Jansen (2011, p.9), revealed that the ecosystem was heavily dependent on five key lone wolves. The results of this analysis confirm the importance of solo developers in software development. The key role played by lone wolves in this community, and any other open source community using a similar approach, certainly calls for a software development methodology for use by these developers. A high-quality software development methodology would therefore enhance the quality of their software products and ultimately those of the ecosystem.

1.11 Limitations of the Study

The SSDM quality framework on which the Secure-SSDM is premised is built on documents obtained through an electronic search. The limitation of this approach is that some unpublished documents or those indexed by databases that were not included in the literature search might have been missed. The quality framework is therefore representative of only those studies that were included in the systematic literature survey. Further, since the methodology is tested through application by an autonomous developer (s), it is not possible to separate the experience or capability of the developer from the quality of the methodology. The quality of a software is dependent on the experience of the team, the methodology in use

and the project environment. Experienced developers can deliver a high-quality product with minimal adherence to a development methodology. Another limitation is that this research did not define any quantitative metrics for evaluating the impact of quality and security practices on the application programmes designed using this methodology. This research used practices that have been proved to be effective by other researchers, therefore proving each practice's effect on the quality of the software of the product is outside the scope of this research.

1.12 Chapter Summary and Thesis Outline

This chapter has highlighted the problem this research is meant to solve. Section 1.6 highlighted the research question and associated objectives, providing answers for each of these. Section 1.7 highlighted the research objectives, showing how each objective was addressed. Further, the chapter gave an overview of the work undertaken in this study. An outline of the research methodology used to build the Secure-SSDM was presented, together with the limitations of the research. The thesis outline is given in the subsequent paragraphs.

Chapter 2 discusses the software development landscape. Starting with the software development history, various achievements in software development are overviewed. This is followed by a discussion of small-scale software development, showing the uniqueness of this environment. An in-depth study of the solo software development environment is undertaken in that chapter to expose the research gap which this research seeks to fill.

Chapter 3 details the research paradigm, research methodology and the data collection methods adopted for the study. Section 3.3 overviews the DSR methodology adopted for undertaking this research. Details of the multiple case study designed to evaluate the utility of the Secure-SSDM are discussed in Section 3.3.5. The theoretical framework used to cement the evaluation of the artefact is also discussed in Section 3.3.5.

Chapter 4 gives an analysis of the SSDM environment, paving way for the formulation of requirements for the Secure-SSDM. Sections 4.3 and 4.4 analyse identified quality and security practices respectively. Section 4.5 discusses the proposed Secure-SSDM's expected quality and security requirements.

Chapter 5 presents the design of the Secure-SSDM. Section 5.2 details the design process as guided by the adapted algorithm of Keramati and Mirian-Hosseinabadi. Section 5.3 gives the details of the stages and activities emanating from the design process. The section concludes by modelling the artefact using the Eclipse Process Framework (EPF) composer.

Chapter 6 discusses the demonstration and evaluation activities carried out to prove the utility of the Secure-SSDM. Section 6.2 details the demonstration of the artefact, followed by the presentation of the academic and industry case study results in Section 6.3 and 6.4 respectively. Section 6.5 gives the theoretical evaluation, followed by discussion of the results. In Section 6.6 threat for validity is discussed.

Chapter 7 reviews the objectives set at the onset of the thesis, showing how these were met. The chapter further gives recommendations for future research, suggesting how other researchers could improve on the practices embedded in the Secure-SSDM.

CHAPTER 2 THE SOFTWARE DEVELOPMENT LANDSCAPE

2.1 Introduction

In Chapter 1, an overview of the work undertaken in this research was presented through detailing the research background, the problem statement, the aim that the research seeks to achieve, research questions and research objectives. In that same chapter, the research methods used to achieve the set objectives and main contributions of the study were overviewed. Justification and limitations of the study were also presented. The chapter concluded by outlining the layout of this thesis.

This chapter provides an answer to the first research sub-question which was stated as:

"**SQ1. What methodologies exist for lightweight solo software development?**

In paving way to provide the answer to this question, a brief overview of the software development landscape in general is given in Section 2.2. This is followed in Section 2.3 by a detail of the very small-scale software development environment. The solo software development draws its characteristics from the latter. Software quality which forms a backbone of this research is discussed in Section 2.4. Reviewing software quality at this stage paves way for the in-depth review of existing SSDMs in the subsequent subsections, where quality practices from existing SSDMs are identified. Section 2.5 presents a systematic review of related work on SSDMs. That section details a meta-synthesis conducted to generate quality theory on existing quality practices on solo software development. In addition, the section presents a quality framework in solo software development. Section 2.6 exposes the security gap in SSDMs by comparing the quality framework derived from existing SSDMs to the ISO/IEEE 25010 (ISO 2010) quality standard. Section 2.7 concludes the chapter by redefining the research direction of the thesis.

Various definitions of software development methodology exist. Pressman and Maxim (2015, p.31) define a software development methodology (SDM) as a systematic approach to software development that guides developers in producing quality software products. These authors use the term software process as a synonym for SDM. In González-Sanabria, Morente-Molinera & Castro-Romero (2017, p.25), a software development methodology is defined as a process organised into a set of phases which offer robust tools and techniques that enable

developers to deliver high-quality software within a defined deadline, and according to set objectives. This definition pertains to an individual development methodology.

The quality of an SDM determines the quality of the resulting software product (Sommerville 2011¸ p.656; Magdaleno et al. 2012, p.1; Iqbal et al. 2016, p.998). Although there are other factors like developer experience, development environment, resource availability, that impact on software product quality, the use of a quality methodology contributes positively to the development of quality products (Fuggetta 2000). To that effect, in pursuit of quality software products, researchers and organisations continue to design high-quality SDMs.

An important dimension of software development is the classification of the development process according to development scale. The scale used differs from country to country, from author to author (Fayad et al. 2000, p.115) and also according to metrics used for the scaling. Common metrics used for scale classification include project time frame, project cost, number of lines of source code, number of requirements and team size (Dingsøyr et al. 2014, p.3). These dimensions are also variable. For example, project costs vary with country while number of requirements vary with type of software product, and number of lines of code vary with programming style, programming language used, and definition of line of code (Marchewka 2015, p.133). Even more, program code could be generated using automated tools (Dingsøyr et al. 2014, p.2), making classification based on lines of code difficult and unreliable.

One way to classify software projects, is to use the number of people in a project. Using this approach, projects can be classified as: very small-scale development (VSD), comprising of one to twenty-five persons; small and medium scale development with more than twenty-five persons but less than two hundred and fifty persons; large scale development, with two hundred and fifty or more persons (Laporte et al 2006, p.3;ISO/IEC 29110 2014, pp.1-3). A broader classification considers fifty or less developers in a project as small scale, and more than fifty, as large scale Fayad et al. (2000, p.115). This research adopts the classification by ISO/IEC 29110 (2014, pp.1-3), since this is an international standard. In this research, the interest is on VSD undertaken by one person. This is referred to as solo development (Pagotto *et al.* 2016, p.2; Ramingwong, Ramingwong & Kusalaporn 2017, pp.342-343).

Studies on software development have concentrated on medium to large scale development, at the expense of very small scale development (Al-Tarawneh et al. 2011, p.1; ISO/IEC 29110 2014¸ p.1; Laporte et al. 2008, pp. 129-130). The design of the ISO/IEC 29110 standard and

agile methods have sought to address this gap. Most agile methods are designed for use by small teams (Boem & Turner 2009, p.28; Schwaber 1997, p.16; Schwaber & Sutherland 2013, p.6). Research however, also shows that despite the focus on small teams by agile methods and the ISO/IEC 29110 standard, very small teams are still using ad hoc processes for software development (Raunak & Binkley 2017, p.3). Further, research also shows that very few studies are focused on solo software methodology design (Dent 2008, p.1; Dzhurov, Krasteva & Ilieva 2009, p.250; Bernabé, Navia & García-Peñalvo 2015, p.687). This chapter investigates research on very small-scale development and poses the following question:

What research has been undertaken in very small-scale and solo software development in view of promoting the software product quality of independent developers?

Before investigating the SSDM environment, it is important to explore the history of software development in general. The history will give the reader the various efforts that have been undertaken in the field, and by so doing show the neglect of the solo development environment. It also provides the reader with trends in the research area, at the same time paving way for new innovations, by drawing ideas from lessons learnt. Reviewing history helps designers to avoid past pitfalls at the same time adopting successes of the past. In the following sub-section, the software development landscape is overviewed.

2.2 Milestones in Software Development

History and the current state of practice in a particular area is important in shaping research efforts (Raunak & Binkley 2017, p.6). Boehm (2006, pp.13-25), provides a ten-year interval starting from 1950 through to 2010. A summary of this progression is shown in Figure 2.1. As shown in the figure, software development practices have evolved from hardware engineering focus (1950s), through code and fix (1960s), through the structured programming era (1970s) which was followed by object orientation (1980s). Object orientation was precursor to agile methods which were introduced around the 1990s. The publication of the Agile manifesto (Fowler & Highsmith 2001) saw the hype of agile methods. An important aspect of this history is the code and fix era which ushered in cowboy programmers. Cowboy (solo) programmers in the 1960s could spend the whole night fixing errors in computer programs for recognition as super-heroes (Boehm 2006, p. 14). This is important in this thesis as it gives us an idea of the origins and characteristics of the solo software development environment. Solo programmers usually do all the development on their own.

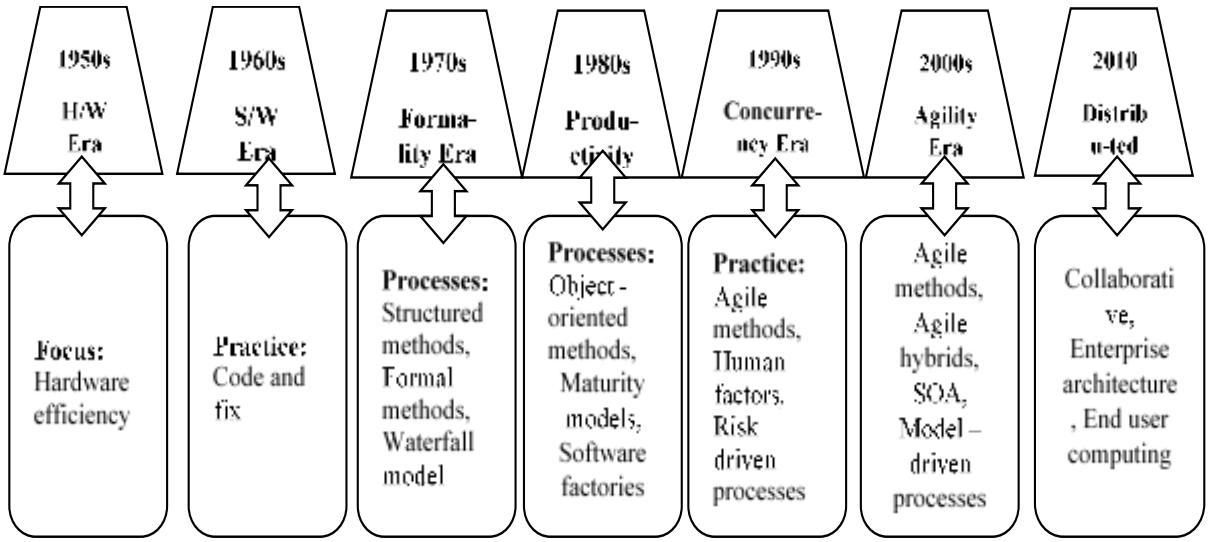


Figure 2.1 Ten-year interval software development landscape (Boehm 2006)

Another highlight in this travelogue is the decade of the 1990s. This decade saw the development of the Personal Software Process (PSP) (Humphrey 1995). Although not a lightweight method itself, PSP is an example of an SSDM. This research derives a lot of influence from PSP. The latter is designed to guide software engineers in the planning and tracking of their development progress (Humphrey 2000, p. 1). Studies on the use of PSP have demonstrated that it improves process and product quality of individual engineers, as well as improve effort and size estimation accuracy (Wesslén 2000, p.122; Pressman & Maxim 2015, p.59; Hayes & Over 1997, p.2). However, despite its positive impact on software quality, its uptake both in industry and academia has been minimal, due to its lengthy training sessions and high training costs (Pressman & Maxim 2015, p.59) as well as its heavy data recording practices (Sison et al. 2005, p.687). These are some of the reasons of undertaking this research.

The same decade saw the advent of agile methods. Agile methods were designed with a focus on small teams (Boehm & Turner 2009, p.28; Schwaber & Sutherland 2013,p.6). Since its origins in the 1990s, agile research and uptake has continued to grow beyond its use by small teams, to large scale and distributed development (Albadarneh 2015, p.1). Raunak and Binkley's recent study shows that agile adoption and research on agile practices is still a

topical issue in industry today (Raunak & Binkley 2017, p.6). However, although having started with a focus on small-scale development, agile research has turned towards large-scale development and distributed agile research. This viewpoint is corroborated by Dingsøyr, Faegri and Itkonen (2014, p.2). Such a move widens the gap between large-scale and very small-scale, and in particular solo software development research. For this reason, this research proposes an agile solo-software development methodology for high-quality software development. In the next section, research on very small-scale development is detailed.

2.3 Very Small-scale Software Development

As cited in Section 2.1, a very small-scale development (VSD) team is made up of one to twenty-five persons (Laporte et al. 2006¸ p.3; ISO 2014, p.1). VSD has been a neglected area of research historically, with more emphasis given to large-scale development (Al-Tarawneh et al. 2011, p.893; ISO 2014¸ p.1; Laporte et al. 2008, pp. 129-130). At the same time these software development organisations contribute significantly to the economies of many countries (Al-Tarawneh et al. 2011¸ p.893; ISO 2014, p.1; Laporte et al. 2017, p.2). Apart from these organisations producing fully developed products, they also contribute important components that are incorporated into large-scale development products (Larrucea et al. 2016, p.85). These components eventually impact on the quality of software produced in large-scale environments. It is important to design processes that promote quality of products created by these organisations, both at component level and full product level (Ayalew & Motlhala 2014, p.49).

The neglect of small-scale development environments (Richardson & Gresse 2007, p.18; Al-Tarawneh Ali 2011, p.1) has led to developers in this environment adapting large-scale methods for their software development projects. This adaptation of methods results in compromised product quality (Pedreira et al. 2007, p.5). Method adaptation is a difficult task that may lead to loss of detail in the adapted method (Ayalew & Motlhala 2014, p.49). Laporte et al. (2006, p.3) demonstrate the difference between small-scale development and large-scale development environments using their priorities in project development. The top ten priorities for each environment are shown in Table 2.1. The colours used here for each practice are meant to assist the reader to locate the priority of a practice in each environment. Priorities that do not match have been left uncoloured (white).

Table 2.1: Top ten development priority list (Laporte, April & Renault 2006)

No.	Small Organisations	No.	Medium to Large Organisations
1.	Managing risks	1.	Consistency among teams
2.	Task estimation	2.	Task estimation
3.	Productivity	3.	Productivity
4	New technology	4.	Team communication
5	Software rework	5.	Process adherence
6	Planning projects	6.	Developing requirements
7	Tracking projects	7.	Ensuring quality
8	Ensuring quality	8	Managing risks
9	Process adherence	9	Managing requirements
10	Maintaining software	10	Tracking projects

Table 2.1 illustrates the difference in priorities between these team sizes. Only six priorities out of their top ten priorities in the list are the same. Although more than fifty percent of the priorities of these team sizes are the same, their emphasis differ remarkably. Only two priorities match at the same level (i.e. task estimation and productivity). Four priorities are ranked differently in the two types of organisations. The medium to large teams' number one priority is consistency among teams. This is not surprising, as the more people in a project, the more difficult it is to coordinate their efforts (Keshta & Morgan 2017, p.570). Knowledge exchange becomes difficult due to the complex communication channels among team members and project sub teams (Schwalbe 2012, p.413). The greater the number of people in a project, the more communication channels needed, slowing down the communication process. Large scale software development processes therefore focus on team coordination

and communication (Dingsøyr et al. 2018, pp.494-495). Team communication in smaller teams is usually direct and therefore not a priority.

On the other hand, small teams' number one priority is risk management. This priority is implicitly addressed in the agile approach (whose target is small teams), where the methods deal implicitly with risks through iteration, daily or weekly meetings as well as onsite customer collaboration (Albadarneh et al. 2015, pp.3-4). While priorities two and three are the same in the two approaches, the rest differ. For example, process adherence ranks as number nine in small-scale development, while it is number five in large scale development. These differences indicate the need for different development practices that address the varying priorities accordingly.

Due to limited resources, small organisations are more concerned with product development than establishing software development processes (Paternoster et al. 2014, p.2). Furthermore, small organisations operate in rapidly changing environments. While the rapid change is not unique to small organisations, such an environment requires that the software development teams regularly undergo appropriate training to keep pace with the changes. Unfortunately small organisations cannot afford regular training programmes due to financial constraints (O'Connor & Laporte 2014¸ p.4; Almomani et al. 2016, p.443). As a result, most of these lose business to highly competitive well-established large organisations (Paternoster et al. 2014, p.1), as these have training programmes to keep their developers up to date with changes in technology.

Over eighty-five percent of software organisations in most countries are small and medium companies (Ayalew & Motlhala 2014¸p.49; Almomani et al. 2016, p.442; Larrucea et al. 2016, p.86; Laporte et al. 2017, p.2). With such a high presence in the market, it is important that these organisations deliver high-quality software in order to attract more customers and retain those that they have (Solyman et al. 2015, p.123).

A number of researchers have explored the VSD environment (Basri & O'Connor 2010; ISO/IEC 2014; Galvan et al. 2015; Wongsai et al. 2015; Larrucea et al. 2016; Laporte et al. 2017; Suteeca & Ramingwong 2017). A study conducted by Basri and O'Connor (2010) to investigate the commitment by very small companies in Ireland to improve their software development methods shows their willingness to the cause. The study also shows that most of the companies participating in the study had adopted agile methods for their development efforts (Basri & O'Connor 2010, p.1450).

Some researchers (Laukkanen et al. 2017; Wongsai et al. 2015) have explored barriers to software process improvement (SPI) initiatives by very small organisations. Suggested as barriers are deployment costs, resource prioritisation and business continuity, among others. It should be noted however that very small organisations stand to enjoy higher financial returns, market recognition, reduced product deployment time if they embraced SPIs such as the ISO/IEC 29110 (Larrucea et al. 2016, p.88). Based on this argument it is important that lightweight SDMs be designed to encourage uptake by very small organisations, in particular solo developers.

In this research a synthesis of quality promoting practices is conducted to derive practices from existing SSDMS to produce a high-quality software development methodology (Pardo et al. 2011, p.95). The research derives quality practices from lightweight methods as these are designed with the solo development environment in mind. Before detailing the derivation of the quality practices from SSDMs, the concept of software quality as a core component of this research is discussed. In the next section the software quality and associated software quality standards are discussed.

2.4 Software Quality

To give a befitting grounding to this research, it is important to explore the subject of software quality. Many definitions of software quality exist (Sfetsos & Stamelos 2010, p.44; García-Mireles et al. 2012, p.134). According to IEEE Computer Society (2014, p.8); "software quality is the degree to which a software product meets established requirements". This definition highlights the importance of stakeholder expectations from the software product, and the importance of understanding those expectations by the developer.

García-Mireles *et al.* (2015, p.150) define software quality from a software product and software development methodology perspective. From the software product perspective, quality is the expected characteristics derived from a quality model to be portrayed by the product, whereas from the software process perspective, quality is the ability of a software development methodology to produce high-quality software products (García-Mireles et al. 2015, p.150). The software product quality perspective requires that with every development effort, the software development team chooses appropriate quality characteristics from a suitable quality model. These characteristics are then used to evaluate the quality of the resulting product. In this case, the chosen quality characteristics form part of the non-

functional requirements of the software product (Nistala et al. 2016, p. 134). The non–functional and functional requirements dictate the conditions of the acceptance of a software product by the user (Kadi et al. 2016, p.1).

Nistala et al. (2016 p.134) define software quality as the ability of a software product to meet user requirements. The assumption here is that user requirements can be determined in advance. Once the requirements are defined, appropriate practices for developing the product are enacted and monitored to attain the required quality. A separate quality assurance team is usually set up to monitor the development team's adherence to expectations. This is a traditional approach to software quality assurance (Marchewka 2015 pp.242-246).

Agile methods have a different approach to software quality. Quality assurance (QA) techniques in agile methods are normally embedded in the software development process (Mnkandla & Dwolatzky 2007, pp.8-9; Sfetsos & Stamelos 2010, p.44; Janus et al. 2012, p.12). Embedding quality practices in the software development method transfers the responsibility of QA to the software development team (Janus et al. 2012, pp. 11-12). This practice ensures that quality aspects are dealt with earlier in the development process, as opposed to validating quality at the end. Characteristically, agile methods shift the QA responsibility to the developers (Huo et al. 2004, p.523). This way the software product is continuously validated and verified as it is being built (Sfetsos & Stamelos 2010, pp.44-45). The embedding of quality practices in the software development process is most favourable for small scale development environments, in particular for solo development environments as this serves as a cost cutting measure.

In pursuing the agile approach to software quality, this research proposes a generic lightweight SSDM that embeds quality practices and techniques to ensure a high-quality software product. A generic SSDM is flexible and can easily be adapted to develop various products (Sutton 2000, p.37). This is appropriate for a solo development environment where resources are limited, and a training budget may not be available to deal with several methods (Basri & O'Connor 2010, p.1456). The researcher defines within the software development process, roles, techniques and practices that support product quality characteristics drawn from the ISO/IEC 25010 product quality model (ISO 2010). It should be noted however that although various roles are defined in the methodology, most of the roles are played by the solo developer, except for the end user roles.

The choice of the ISO/IEC 25010 quality model as a reference point for product quality was inspired by other researchers (such as Suryn 2014 p.51; García-Mireles et al. 2015, pp.150-166; Kadi et al. 2016, pp.1-8; Nistala et al. 2016, pp.144-147; Idri et al. 2017, pp.262-267) who have used the model as a quality reference in similar projects. Further, as an international standard, the model facilitates benchmarking of the developer's products with those of the rest of the world (Galvan et al. 2015, p.189). By using comprehensive quality techniques to embed quality in the SDM, the research eliminates the need for a project management methodology and a separate quality assurance team. This is a cost cutting measure for a solo development environment, where financial resources and resources in general are limited. The proposed methodology therefore assists in cutting costs associated with the establishment of a separate quality assurance team.

2.4.1 Software Quality Models

Software quality models offer a systematic approach to defining quality requirements, building the required quality into the product and monitoring the quality process (Wagner et al. 2015, pp.102-103). A software quality model provides a way of breaking down abstract quality concepts into measurable concrete terms (Lew 2012, p.2). Quality models provide a basis for specifying quality requirements of a product under development as well as evaluating the specified quality (Suryn 2014, p.14). Traditionally, software quality models are tools used to portray the interaction between various quality factors. These factors are usually grouped into high and low-level factors. High level factors are abstract and what we desire to measure, whereas low level factors are more concrete and understandable providing means for measuring the high-level factors.

The first examples of quality models included those of McCall, Richards and Walters (1977), Boehm's model, the functionality, usability, reliability, performance, and supportability (FURPS) model and the ISO/IEC 9126. With changes in the computing environment, the ISO/IEC 9126 has since been revised to the ISO/IEC 25010 model, the chosen model for this research. Figure 2.2 shows the ISO/IEC 25010 product quality model. The ISO/IEC 25010 quality model defines guidelines for defining and evaluating software quality requirements (Kadi et al. 2016, p.1). As shown in Figure 2.2, the centre part illustrates the quality characteristics defined by the model, while the extreme right shows the measurable sub - characteristics of the product. To illustrate the interpretation of this figure, consider functional suitability as a characteristic of quality. A functionally suitable software product should

portray functional completeness, correctness and appropriateness. These are the measurable sub-characteristics. Based on the definition of these sub-characteristics, appropriate metrics and ranges can be defined and used to measure these sub - characteristics which in turn give measures for the characteristics.

The models highlighted here so far are known as definition models. Definition software quality models describe quality characteristics to be portrayed by a quality product, but they do not necessarily define how to build these characteristics into the product (García-Mireles et al. 2015, p.150).

In this research, the ISO/IEC 25010 quality model was chosen as the model to base quality on. The researcher concurs with García-Mireles et al. (2015, p.151) that this is a good model to use as a base to develop software products. The model classifies quality into software product quality and quality in use. The research identifies quality promoting practices in existing SSDMs and maps these to quality characteristics defined in the model. In doing so the research posits that, existing methodologies have quality practices that can promote the building of quality characteristics defined in this model. These practices can be identified and synthesised to design a higher quality software development methodology. A meta–synthesis is conducted on existing SSDMs to identify those practices that support quality characteristics defined in this model. This study is therefore similar to that of García-Mireles et al. (2015) in that it determines the support of existing software development methods for the product quality characteristics of the ISO/IEC 25010 model. It however differs in that whereas these authors looked at software process improvement versus the quality model, here the researcher looks at SDMs, in particular quality practices in SSDMs versus the quality model.

The synthesis of the quality practices from a number of SSDMs is considered important in methodology design as it ensures that a higher quality methodology than the component methodologies is produced (Pardo et al. 2011, p.95). To ensure a systematic mapping of the practices extracted from the methodologies, to the characteristics of the ISO/IEC 25010 model, first the researcher identified themes from those practices in participating SSDMs, and compared them with the model characteristics. The product quality characteristics used for comparison are functional suitability, performance efficiency, compatibility, reliability, usability, maintainability, security and portability (ISO 2010). Adopting a product focused quality approach ensures quality practices are built into the methodology to deliver the product quality defined in the model (Trienekens et al. 2002, p. 269). Modelling a methodology around

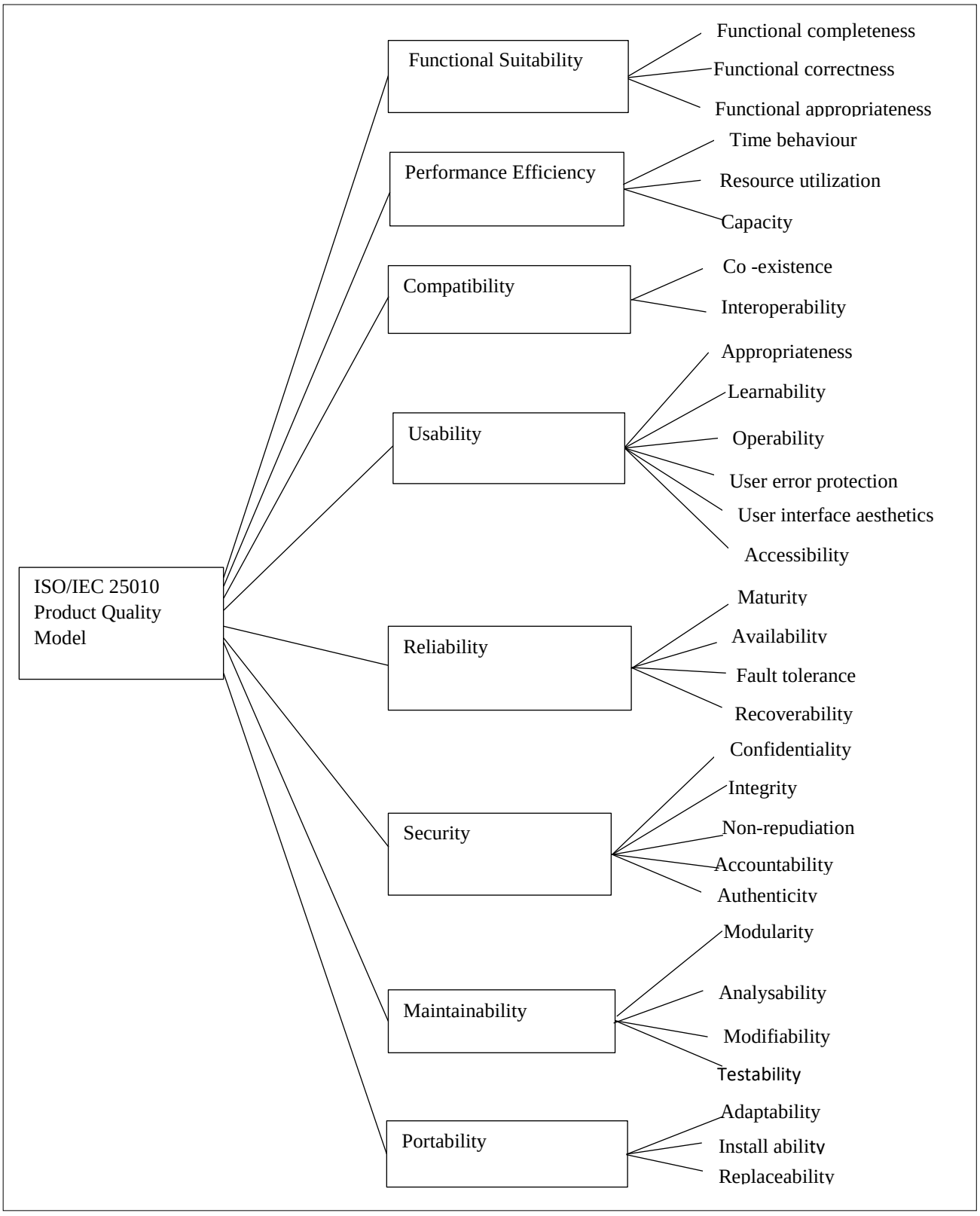


Figure 2.2 : ISO/IEC 25010 quality practices (ISO 2010)

a quality model makes it flexible as developers should be able to implement relevant practices based on the quality requirements of the software product at hand (Pedreira et al. 2007, p.1).

2.5 Review of the Solo Software Development Environment

As seen in section 2.2, Solo software development (SSD) dates back to the 1960s during the code and fix era where cowboy programmers spent the whole night fixing errors in code (Boehm 2006, p.14). The introduction of multiprocessing operating systems introduced team development, shifting the focus to large-scale software development. This shift has side-lined solo software development (Hollar 2006, p.1; Dent 2008, p.1; Dzhurov, Krasteva & Ilieva 2009, p.250; Abrahamsson et al. 2013, p.6). A large number of software products in the market today is developed by micro teams. A micro-team is a team of one to two developers (Ramingwog et al. 2017, p.342). To ensure high-quality products, these solo developers adopt and adapt the available small-scale or large-scale methodologies for their development efforts. Methodology adaptation if not properly done in some cases leads to loss of detail, thereby compromising the quality of the resultant methodology (Pedreira et al. 2007, p.5; Ayalew & Motlhala 2014, p.49). The solo development environment is unique in that it exhibits the following characteristics , which are inherited from VSD (Laporte et al. 2006, p.3; ISO/IEC 2014, p1) :-

i. Limited resources – where the developer is the sole owner of the development house, resources tend to be limited (Wongsai et al. 2015, p.14; Keshta & Morgan 2017, p.24163). The available resources are channelled towards the actual development effort, and rarely on software project support, such as training and documentation (Coleman & O'Connor 2008¸ p.773; Basri & O'Connor 2010, p.1457).

ii. Lack of historical data – Due to limited resources, solo developers may not have the capacity to maintain a database of past projects (Paternoster et al. 2014, p.2). This makes effort and resource estimation a difficult process to execute in project management (Sommerville 2011, p.636).

iii. Fast development time – The current software development environment demands fast software product delivery. Apart from dealing with fast development speed that characterises today's software industry in general, solo developers need to deal with the execution of simultaneous projects for survival in the market (Bernabé, Navia & García-Peñalvo 2015, p. 687).

Such an environment requires the use of SDMs specifically designed to address these characteristics (Coleman & O'Connor 2008, p773; Basri & O'Connor 2010, p.1457). Some

researchers (Humphrey 2000; Agarwal & Umphress 2008) have tackled this problem by developing software processes specifically targeted at this environment. The Personal Software Process (PSP) (Humphrey 2000) is widely accepted by both industrialists and academics as an SDM designed for individual developers (Dzhurov, Krasteva & Ilieva p.252; Abrahamsson et al. 2013, p.2; Pressman & Maxim 2015, p. 59). If properly applied, the model helps engineers to systematically plan their work, using their personal data from previous performance (Pressman & Maxim 2015, p. 60). PSP enables consistent improvement on developer performance, as well as the production of quality software products through identification and fixing of defects early in the software process (Humphrey 2000, p. 24; Abrahamsson et al. 2013, p.3).

While PSP ensures that quality is built into the development process, and subsequently into the product, its main problem is that it is document heavy. Developers using PSP spend so much time collecting and documenting their progress, instead of developing the actual system (Dzhurov, Krasteva & Ilieva 2009, p.252). Due to its heavy documentation, complexity, lengthy training sessions and high training costs, PSP has not been widely adopted in industry. In cases where PSP is used, just parts of the method are utilised (Pressman & Maxim 2015, p.60). The excessive documentation associated with PSP and its high training costs, give this study the urge to design a lightweight and low cost SSDM.

In designing the methodology, the few existing lightweight SSDMs are viewed as a knowledge base of best practices designed to address the unique characteristics of the solo development environment. Further, the research posits that, using an appropriate methodology, the quality practices in this knowledge base can be synthesised to produce a higher quality SSDM (Peffers et al. 2008, p.49). For this part of the literature review, the research adopts a qualitative approach to identify and synthesise practices from published research on SSDMs. The aim of the review is to derive a quality theory for solo software development.

2.5.1 SSDM Meta-synthesis

Meta-synthesis is a systematic way of building knowledge from existing literature. It enables the researcher to make use of existing knowledge in creating new knowledge. Meta-ethnography (Noblit & Hare 1998) is one example of a knowledge synthesis approach used in meta-synthesis to build theory from qualitative studies (Runeson *et al.* 2012, p.117). In this

review, meta-ethnography is used to synthesise the quality practices from existing SSDMs as it systematically facilitates the derivation of theory from existing data. Meta-ethnography is preferred as it enables a systematic study of the existing methods through comparing concepts in and across the studies (Napoleão & Rodrigo 2018). Further, some researchers (such as Siau & Long 2005; Napoleão & Rodrigo 2018), have used the method in deriving new methods from existing ones.

Meta-ethnography enables the researcher to derive themes from existing methodologies so as to build a stronger theory (Cruzes & Dybå, 2011, p.443). It also helps in identifying gaps in the quality practices in SSDMs (Mohammed et al. 2016, p.696). With minimal published research in SSD, meta-ethnography is the most appropriate as it does not necessarily need a large number of studies for the synthesis (Noblit & Hare 1998, p. 111). Exploring the existing SSDMs and the different practices that those methodology designers have integrated into their methods provides an insight into the norms in methodology design (Stewart et al. 2012, p.342). Sub-section 2.5.2 details how the guidelines given by Noblit and Hare (1998, pp.109-113) and Mohammed et al. (2016, pp. 697-699), were used to conduct the meta-ethnography.

2.5.2 Conducting the meta-ethnography

In conducting the meta-ethnography, the following stages as defined by Noblit and Hare (1998, pp. 109 – 113) were adopted: -

(1) Getting started - this entails choosing a topic of interest to the researcher (s) that could benefit a set of practitioners. A research question is usually defined to represent the topic and serve as a guide in the meta-ethnography process.

(2) Choosing the relevant studies – entails selecting studies that fall under the defined topic. This is done through searching for the studies in relevant sources and defining inclusion and exclusion criteria for selecting the studies.

(3) Reading the studies – involves repeatedly reading the articles to understand the content of the participating studies. Data extraction begins at this stage with researchers extracting the main points from the studies.

(4) Determining studies relationships – this can be done through creating a list of the key metaphors from each study. Once the metaphors from each study are created, tables or grids can be used to determine the relationships among the key concepts.

(5) Translating the studies into each other – Metaphors from the participating studies are compared to each other. This may be done through listing metaphors from the first study, and comparing each of the metaphors in the participating studies with those of the first study.

(6) Synthesizing translations – involves grouping common metaphors and in some cases subsuming metaphors in others. Diagrams may be used to represent the relationships among the metaphors.

(7) Expressing the synthesis – at this stage appropriate channels are used to disseminate the findings of the meta-ethnography to the intended audience.

The following paraphs detail how these steps were used in this meta-ethnography.

Step 1: *Getting started*

The researcher established the following research questions to guide the meta-synthesis:

Question: -

How do current Solo Software Development Methodologies enable quality in the developed software?

The related sub-questions were:

1. *What methodologies exist for solo software development?*
2. *What practices and techniques are used to ensure the production of high-quality products in these methodologies?*
3. *How do the identified practices and techniques enable quality in the final product?*
4. *What theories emerge from current solo software development practices?*

Step 2*: Searching for relevant studies*

The researcher conducted a search on databases and journals publishing Software Engineering research. The search was conducted from December 2017 to April 2018. The list of databases and journals chosen in this research indexes Software Engineering publications. The list has also been used in part or in full by many researchers (for example, Dybå and Dingsøyr 2008, p.6, Sfetsos and Stamelos 2010, p.45, Selleri Silva *et al.* 2015, p.23 and Zarour *et al.* 2015, pp.181-182) on similar reviews. These sources are also suggested by Brereton et al. (2007, pp.577-578) as appropriate for Software Engineering literature surveys. The sources are:

ACM Digital library; Scopus; ScienceDirect; INSPEC; ISI Web of Science; SpringerLink and IEEE Xplore. Google Scholar was also used to search the World Wide Web to ensure all articles describing solo software development methods were identified.

The search string used with each of the data sources was derived from the main research question, and is given below. On searching each of the sources, the string was adjusted according to the defined syntax in the database, taking care to maintain the meaning of the string.

> *("software development methodology" OR "software process" OR "software process model") AND ("solo" OR "freelance" OR "independent developer" OR "autonomous" OR "personal") AND ("quality").*

From the retrieved studies, selected studies for the synthesis were based on the following inclusion criteria: -

a) Only papers published between January 2000 and December 2017 were included. This period coincides with the hype of agile methods, whose focus is small scale development, and are light weight.

b) Only publications written by the author of the methodology are included. This enabled the researcher to get first-hand information from the publications.

c) Only publications describing lightweight solo development methods were included. This is in line with Sandelowski, Docherty and Emden (1997, p. 368)'s advice to screen studies according to "topical similarity."

Exclusion criteria were as follows: -

a) Documents discussing a methodology of team size of more than one,

b) Documents by a second author describing another's methodology

c) Documents comparing any software development methods and

d) Tools used to automate software development methodologies.

The retrieved number of articles according to database is shown in Table 2.2. For the purposes of screening the articles, these were exported to Microsoft Excel so that the documents could be easily processed.

Table 2.2: Database search results

Database	Search Results & Duplicate Screening	Title Search Elimination	Abstract Elimination	Articles Included
ACM Digital Library	2072	2056	14	2
IEEE Xplore	273	265	8	0
Scopus	67	63	1	3
Science Direct	812	809	3	0
ISI Web of Science	35	33	0	2
SpringerLink	202	201	0	1
INSPEC	245	242	3	0

As shown in Table 2.2, the search against the ACM digital library identified two thousand and seventy-two studies published between 2000 and 2017. From this database, sixteen were eliminated through duplicate screening, leaving two thousand and fifty-six studies. Duplicate screening is easier in MS Excel through the use of the 'Remove duplicates' function. Two thousand and forty-two were eliminated through title scrutiny, to remain with fourteen. After reading the abstracts of the fourteen studies, twelve were eliminated leaving two studies. The information from other databases is interpreted similarly. Five articles in total were found the digital libraries. Three articles appeared in more than one digital library. Go-Scrum appeared in Scopus and SpringerLink, while Faat appeared in Scopus and the ACM digital library. Scrum Solo appeared in Scopus and ISI Web of Science. A search on Google Scholar led to the identification of a sixth publication, DeSoftIn. The six articles that survived abstract screening were deemed suitable for the synthesis. The six studies provided the answer to the first research question:

1. *What methodologies exist for solo software development?*

The methodologies retrieved through our literature search are:

i. Freelance as a Team (Faat) (Bernabé, Navia & García-Peñalvo 2015),

ii. Personal Extreme Programming (PXP1) (Agarwal & Umphress 2008),

iii. Personal Extreme Programming (PXP2) (Dzhurov, Krasteva & Ilieva 2009)

iv. Go – Scrum (Ramingwong, Ramingwong & Kusalaporn 2017)

v. Scrum Solo(Pagotto *et al.* 2016) and

vi. DeSoftIn (González-Sanabria, Morente-Molinera & Castro-Romero 2017).

Numbers here were used to differentiate the two PXPs. It should be noted that this answer helps this research to provide the answer to the first sub-question posed in this research posed as:

SQ1. What methodologies exist for lightweight solo software development?

While a publication in the year 2000 of the PSP by Watts Humphrey was retrieved by the search, it was not included in the analysis as it is a heavy weight methodology. It was excluded using the inclusion and exclusion criteria.

Step 3: *Reading and Re-reading the Selected Literature*

All peer reviewed articles retrieved from the databases selected for this synthesis and meeting the inclusion and exclusion criteria defined in step 2 were considered to be of acceptable quality for this research (Sandelowski et al. 1997, p.368). All the six studies were included in the synthesis. Using a pre-prepared extraction template (Table 2.3) premised on the studies (Mohammed et al. 2016, p.697), the data from the publications was extracted. The table format ensured that all concepts from the authors are extracted (Cahill et al. 2018, p. 133). Each methodology name was captured together with the author and year of publication. The methodology stages and quality practices of each stage were entered into the second column of the table. The third column shows how each practice contributes towards quality in the developed software. This is based on the interpretation of the author of the methodology.

During data extraction, the publications were read several times in full and the researcher extracted the data during the reading. The researcher ensured data extraction accuracy by iterating though the stages of the meta-ethnography process, checking extracted data against original documents at every stage.

Table 2.3: Data extraction template

Title & Author	**Quality Practices/ Techniques**		**Quality characteristic promoted in the final product**
	Stage (s)	**Technique(s)**	
1. Freelance as a Team (Faat) (Bernabé, Navia & García-Peñalvo 2015, p.687-694)	STRATEGIC PRACTICES		
	Simplicity Application of minimum viable product and minimum marketable features techniques		Promotes testability, understandability, browsability and, system explain ability (p. 687) Reduces development time

	Embrace Change Establish points of stable code, fix bugs early, use product versioning		Promotes code failure recovery & product completeness
	Making Decisions Stick to specified requirements, avoid gold plating		Reduces development time
	DEVELOPMENT STAGES		
	F1. Knowledge and Motivation	Learning the methodology	"Equips developer with project management processes." (p. 691)
	F2. Preparation of backlog	Creation of a product backlog	Promotes project & product completeness
		Formulation of small tasks	Enhances user acceptance
		Prioritisation of tasks	
	F3. Creation of User Stories	Generation of small story cards	Promotes requirements completeness & product simplicity
	F4. Estimation (Iteration start)	Comparison of actual & estimated times at iteration end	Promotes time estimation accuracy
	F5. Planning	User story prioritisation	Promotes end user acceptance
		Definition of internal & external deliveries to form cycles	Promotes product simplicity
		Refactoring of big stories	Promotes system understandability Promotes developer motivation
		Setting of short iteration duration (2 -3 weeks)	"
		Respect of cycle times	
	F6. Development	Use of version control for all code	Promotes code traceability
		Creation of test cases for all code at start of user stories	Promotes defect reduction "
		Documenting tested code	
	F7. Review	Performing of code coverage tests	Promotes defect reduction & code quality
		Review of technical debt	Promotes code quality

		Review of code using a rubber duck	Enhances code quality
		Class dependency & maintainability checks	Promotes product maintainability Promotes design quality
		Performing simplicity checks	
	F8. Iteration Close	Use of version control systems	Promotes code quality &product compatibility; Eases return to the last stable code
	F9. Evaluation Continuous practice, runs in parallel with all the practices	Evaluation of software quality	Allows for process improvement and refinement
		Evaluation of software performance	Improves system performance
		Evaluation of development process	Improves development methodology quality Promotes component reusability
		Identifying processes for automation	
	AUXILLARY PRACTICES		
	Refactoring		Minimises code smells and anti –patterns
	Minimal documentation		Reduces development time
	Planned partial prototyping		Promotes user requirements clarity
	Use of a dummy partner (rubber duck)		Promotes code quality
	Task automation		Promotes task reuse & eases development effort
2. Personal Extreme Programming (Agarwal & Umphress 2008)	*P1.1 Start*	Adoption of a coding standard	Promotes product consistency
	P1.2Planning	Requirements statement using: -Metaphor -User stories	Promotes user requirements understanding
		Creation of features from user stories	Promotes design simplicity
		Creation of domain design	“
		Prioritisation of features	“
		Size & Time estimation	Reduces schedule risk
		Use of design acceptance tests	Ensures focus on product
		Creation of iteration schedule	Promotes development speed

	P1.3 Development	Product feature prioritisation	Promotes user participation & acceptance
		Breakdown of features into tasks	Promotes development simplicity
		Creation of task priority list	Promotes development speed
		Creation of task unit tests	Promotes product quality
		Performing code walkthrough	Enhances code quality
		Practicing version control	Promotes code consistency
		Performing acceptance tests	"
		Code Integration	"
		Use of iteration releases	Promotes early product release & user acceptance
	P1.4 Post Mortem	System acceptance test	Promotes system acceptance & product quality
3. Personal Extreme Programming (Dzhurov, Krasteva & Ilieva 2009)	*P2.1 Requirements*	Adoption of design and coding standards	Promotes development consistency
		Creation of requirements list	Promotes product completeness
	P2.2 Planning	Breakdown of requirements into tasks & subtasks	Promotes development simplicity
		Categorisation of subtasks	Promotes development speed
	P2.3 Iteration initialisation (1 – 3 weeks)	Task prioritisation	Promotes early delivery of core tasks
	P2.4 Design	Design of system modules	Promotes product simplicity
		Design of classes	"
	P2.5 Implementation	Use of coding standards	Promotes product quality
		Testing of modules (units)	"
		Refactoring code	"
	P2.6 System testing	Checking system against user requirements	Promotes user acceptance
		Early fixing of errors	Promotes defect reduction
	P2.7 Retrospective	Analysing developer	Determines improvements on performance

		performance in phases	
		Checking actual against estimates	Promotes estimation accuracy Promotes timely delivery
		Release of product in components	
4. Go-Scrum (Ramingwong, Ramingwong & Kusalaporn 2017)	*G1. Management Buy-in*	Development process explanation	Encourages user participation & product acceptance
	G2. Kick-Off Meeting & Story Discovery	Meeting with users	Promotes user participation Encourages requirements understanding
		Use of user story cards	
	G3. Project Planning	Creation of product backlog	Promotes user acceptance
	G4. Release & Sprint Planning	Product backlog prioritisation	Promotes development transparency Promotes development speed “
		Product backlog time estimation	
		Creation of a sprint backlog	
	G5. Sprint	Sprint review	Encourages communication between developer and users, Promotes development speed
		Sprint retrospection	
		Sprint planning	
5. Scrum Solo (Pagotto *et al.* 2016)	*S1. Requirements elicitation*	Scope definition	Promotes product completeness
		Customer identification	Promotes user acceptance Promotes product completeness
		Creation of product backlog (software requirements)	
		Prototyping	Facilitates user requirements understanding
		Use of a data Repository (stores scope, product backlog and product prototype)	Promotes communication with users
	S2. Management *(Overarching activity, initiated at Sprint onset)*	Use of Gantt charts in planning	Promotes development speed Promotes product completeness Promotes project management Promotes development speed “
		Use of a WBS	
		Size & budget estimation	
		Monitoring & control of time	
		Review of project progress	
	S3. Sprint (1 week)	Use of Sprint backlog	Promotes product completeness
		Creation of development plan	Promotes development speed

		Recording of time and effort estimates	“
		Coding with code review	Promotes defect reduction
		Testing	Promotes code quality
	S4. Deployment	Product validation	Promotes user acceptance
6. DeSoftIn (González-Sanabria, Morente-Molinera & Castro-Romero 2017)	*D1. Planning and analysis*	Setting of project scope	Promotes product completeness
		Identifying customer financial capabilities	Enables definition of scope
		Defining & prioritising sprint activities	Promotes development speed
		Use of a colour coded requirements checklist	Promotes product completeness & Visualises development progress
		Use of short development sprints (3 – 10 days)	Facilitates product changes & development visibility; reduces product risk
		Taking breaks between sprints	Promotes independent self-criticism; facilitates knowledge acquisition
		Use of a diary (log book)	Promotes progress tracking
	D2. Design	Use of design modelling tools	Promotes understanding of business environment
		Creation of system prototypes	Promotes product verification
		Use of Class responsibility collaboration cards	Promotes design completeness
	D3. Development	Iterative delivery	Promotes user acceptance & development speed
		Use of a colour coded development checklist	Promotes development transparency & speed
		Self-criticism	Promotes product quality
	D4. Implementation	Module implementation & integration	Promotes product maintainability
		Module validation	Promotes product quality
		Module integration testing	Promotes product quality
		Use of quality & security standards	Promotes product quality

		Use of risk management strategies	Minimises project failure
	D5. Evaluation	Checking of adherence to user requirements	Promotes product acceptance
		Meeting with consultant	Enhances developer technical knowledge/ skills & enhances product quality

Table 2.3 provides the answer to the second and third questions posed for the literature review as follows:

2. What practices and techniques are used to ensure the production of high- quality products in these methodologies?

The practices and techniques in the third column of Table 2.3 promote quality in the developed software. As shown in the table, the practices are organised to promote quality in each stage of the development process as defined in the methodology. Using the last entry in the table, developers adopting DeSoftIn as a methodology end with an evaluation stage. Quality practices at this stage entail checking of developer adherence to user requirements and arranging a meeting with the consultant.

Since DeSoftIn is designed for use in an academic setting, consultancy is readily available. At the end of a development cycle, the academic supervisor sits with the student developer to check adherence to the development process. Other practices in the table are interpreted similarly.

3. How do the identified practices and techniques enable quality in the final product?

Similarly, to answer this question, using the same example of the last entry in DeSoftIn, checking developer adherence to user requirements promotes user acceptance. At the same time having a meeting with a consultant at this stage to evaluate the just ended sprint or project enhances developer skills, which in turn improves product quality. The impact of the other practices and techniques are also interpreted the same way.

Step 4: *Determining Relationships among the Studies*

The data extraction template in Table 2.3 was used to derive the relationship among the methods through capturing of the key concepts (Mohammed et al. 2016, p.697). Stages of

each methodology were extracted together with quality practices in each of the stages. Looking at the six methods, there are some common stages and practices among all the methodologies. For example, all methodologies have a Planning, Development and Evaluation stage. Although these are named differently in the various methods, the software development activities in these are similar. All methodologies emphasise the creation of a product backlog at the onset of development. In PXP2 (Dzhurov, Krasteva & Ilieva 2009, p. 254) this is called a requirements document, while this is termed a feature set in PXP1 (Agarwal & Umphress 2008, p.83). A closer look at the two PXP methods shows that they share a lot in common as they are both hybrids of PSP and Extreme Programming (XP). The difference between the two is that PXP2 assumes that requirements can be identified, prioritised and fixed at the onset of the project, with changes in the environment calling for change in task re-prioritisation. PXP1 and all the other methods accommodate requirements change throughout the project.

Go - Scrum (Ramingwong, Ramingwong & Kusalaporn 2017) and Scrum Solo (Pagotto *et al.* 2016) also share a number of characteristics drawn from Scrum. Go – Scrum defines a stage, Management Buy-in, to encourage methodology acceptance in a bureaucratic environment. This is a unique feature of this method among the six methods considered in this study, perhaps due to the fact that it was designed for use in a government environment (Ramingwong, Ramingwong & Kusalaporn 2017, p. 343). Scrum Solo has a cross life cycle activity, Management, with practices that can be used at any of its stages. Its Management practices are similar to the Strategic practices in Faat in that they are applied on demand at any of the methodology stages. Faat defines three stages Knowledge and motivation, Implementation and Evaluation. Implementation is made up of a number of sub-stages (Prepare product backlog, Creation of user stories, Estimation, Planning, Development, Review and Iteration close) (Bernabé, Navia & García-Peñalvo 2015, p.691). These have been indicated as stages in Table 2.3 to allow for ease of comparison with other methods. These six methods have a lot in common enabling their translation into each other (Noblit & Hare 1998, p.111).

Step5*: Translating Studies into each other*

Using recommended translation approaches (Noblit & Hare 1998, p.111; Mohammed et al. 2016, p.698), the six methodologies were translated to each other to facilitate the generation of a quality theory. A template drawn from the data in the studies was used to produce the

translation depicted in Table 2.4. Faat was used as a template as it has the highest number of stages (nine), and is more detailed. The ninth stage, Evaluation is a cross life cycle activity executed simultaneously with each stage to assess methodology efficiency (Bernabé, Navia & García-Peñalvo 2015, p.693). In translating the studies, each method was compared against Faat, and similarities and differences noted. The first method considered is PXP1. PXP1 has four stages, Start, Planning, Development and Post Mortem.

Table 2.4: Translation of studies

Stage	Faat	PXP1	PXP2	Go – Scrum	Scrum Solo	DeSoftIn
I. Knowledge & Motivation	√	Start	Requirements	Management Buy-in		
II. Preparation of Product backlog	√	Planning	Requirements	Kick –off Meeting & User story	Requirements	Planning & analysis
III. Creation of User Stories	√	Planning	Planning	Project Planning	Management	
Iteration initiation	√	Planning	√	Release & Sprint Planning	Sprint & Management	
Planning	√	√	Design			Design
IV. Development	√	√	Implementation	Sprint with Inspection	Sprint & Management	Development
V. Review	√		System testing			Implementation
Iteration Close	√		System testing	Sprint with Inspection	Deployment & Management	
VI. Evaluation	√	Post mortem	Retrospective			Evaluation

The Planning stage in PXP1 consists of user story elicitation, creation of a feature list and prioritisation of the list. This is similar to the Preparation of backlog, Creation of user stories, Estimation, and Planning stages of Faat. Due to the similarities in these stages, they can be translated into each other. The Development stages are the same, although Development in PXP1 entails code review, acceptance testing and iteration release, which are activities pushed down to a different stage called Review in Faat. These were therefore put in the appropriate stage.

PXP2's first stage, Requirements, is similar the first stage of PXP1 called Start, in that at both stages the developer adopts design and coding standards for use in the development process. The two methods map directly into Faat's Knowledge and Motivation stage as here the developer learns the methodology and all activities to go with the method. PXP2 is unique in that it separates the stages Design and Implementation. This concept of PXP2 is similar to the approach used in DeSoftIn. However if the developer upholds simplicity advocated for by Bernabé, Navia and García-Peñalvo (2015, p.687) these two stages can be combined and be executed as in Go – Scrum. The colour codes in Table 2.4 show how the different stages can be mapped onto each other. Scrum Solo is the only methodology that does not suggest an initial stage where the developer takes time to learn the methodology for use. The learning of the methodology in DeSoftIn is suggested to be done during sprint breaks. Here the developer is advised to consult an adviser in the field who can check the developer's adherence to the adopted methodology, and suggest means for improvements as necessary. This is a unique feature of this methodology in that it assumes the availability of a ready consultant, since it is developed for an academic setting. The other methods have the initial stage dedicated to adoption of standards and understanding of the method.

The translation of the stages into each other has helped the researcher to discover the underlying themes on quality practices from individual studies enabling the construction of a comprehensive framework that advances knowledge in quality supporting techniques in solo software development (Siau & Long 2005, 449). This framework, as an abstract model enables the understanding of what is currently prevailing and serves as a basis for the formulation of a richer method (Gherib et al. 2015, p. 420).

Step 6: Data Synthesis

In synthesising the data, this research uses the translations of the studies in step 5 to bring together the identified themes so as to derive meaning from the data. The research used guidelines for the translations as suggested by (Seaman 1999, p.568). The data from the various methodologies was compared iteratively. First the quality concepts from Faat were extracted as shown in Table 2.3. These concepts were analysed for quality promotion. Next the concepts from PXP1 were compared to the concepts in Faat. Similarities and differences among concepts were noted. Similar concepts were consolidated and different concepts from PXP1 were added to the list of concepts drawn from Faat. Propositions were generated based on the concepts from the two methods. Next the concepts from PXP2 were considered and

mapped against the concepts from the already existing propositions. New propositions were added in cases where there were no matching propositions in place. In some cases, propositions were modified to accommodate concepts from PXP2. The remaining three methods were synthesised similarly. To enhance validity of the synthesis, and theory generated thereafter, maximum effort was made to support all derived propositions from the studies with references (Mohammed et al. 2016, p.698).

Since the main interest in this research is to use existing methods as a base for the proposed methodology, the stages and practices in the methodologies were grouped into stages as shown by the map in Table 2.4. Codes have been adopted for ease of illustration. As an example, the stage codes S1, S2, S3 and S4 correspond the stages of Scrum Solo; Requirements elicitation, Management, Sprint and Deployment respectively. Note that the stage, Management in this methodology is a cross life cycle activity, since the developer reviews progress at every stage of the development (Pagotto *et al.* 2016). Practices and techniques used in each of the stages were analysed to establish the relationships among them. The synthesis was mapped to stages so as to derive theories within the stages. Figure 2.3 illustrates the grouping of activities within the stages to facilitate stage by stage theory derivation. The activities from the methodologies were grouped into six stages representing the proposed developmental process. While the interest of the synthesis is on identifying emerging theories on quality practices and how they support quality in the ultimate product, the grouping of these practices into stages helps the researcher to understand how these practices would support product quality in these stages. The ultimate goal in this research was to build a solo software development methodology that supports the delivery of high-quality products. Therefore, the grouping of activities into stages enables this thesis to propose a framework for the development of a new methodology. The framework is discussed in the following sub-section.

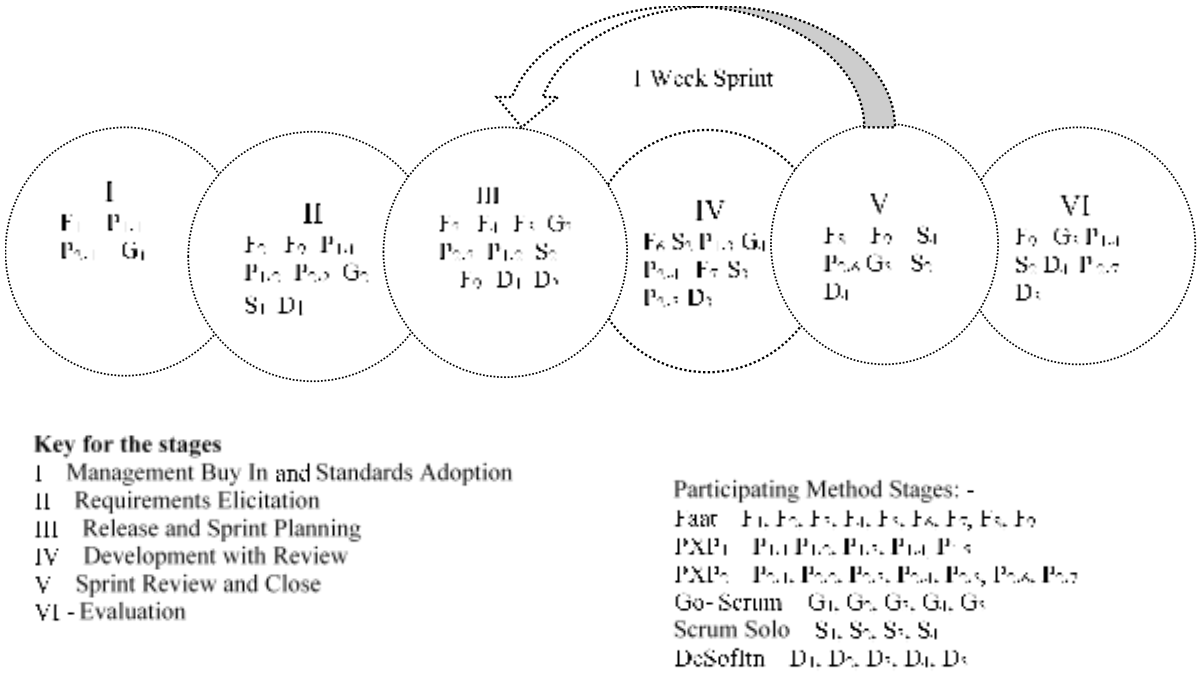


Figure 2.3 : Grouping practices in the SSDM framework

2.5.3 The Secure-SSDM Primary framework

The meta-synthesis enabled this research to formulate a primary framework for the proposed methodology. The stages I to VI summarise the activities derived from the synthesis that would subsequently promote quality in the developed software product.

Stage I: Management Buy-in and Standards Adoption

The first stage in the derived framework is a familiarisation stage, where the developer learns the process and adopts appropriate software development standards. The concept of adoption of standards at the onset of the project is drawn from the practices in the first stages of Faat (Bernabé, Navia & García-Peñalvo 2015), PXP1 (Agarwal & Umphress 2008), PXP2 (Dzhurov, Krasteva & Ilieva 2009), and Go- Scrum (Ramingwong, Ramingwong & Kusalaporn 2017). Go-Scrum includes a unique stage, Management Buy –In, with a practice of educating the stakeholders on the method used to develop the software product. This practice is very important in a solo environment and in software development in general. If properly executed, it enhances user participation in the development process, as users get to learn how software development will proceed at the onset of the project.

The practices have been added in the first stage of the framework since user participation in general promotes user acceptance of the product at the end of the project (Ramingwog et al. 2017, p.344). This first stage of the framework has been termed Management Buy-in and Standards Adoption. The standards adopted at this stage guide the developer towards the development of a quality software product. Management Buy-in and Standards Adoption captures all the practices related to the environmental management of the development process.

Three propositions emerge from this stage:

i. *Educating users on the methodology to be used in the development of the project, facilitates user participation which enhances user acceptance of the software product* (Ramingwog et al. 2017, p.343).

ii. *Adoption of developmental standards at project onset encourages development consistency by the developer* (Agarwal & Umphress 2008, p.85).

iii. Early user involvement promotes user participation and facilitates product acceptance (Ramingwong, Ramingwong & Kusalaporn 2017).

Regarding the meeting held during the Management Buy-in (Go – Scrum), the authors consider the practice as important since according to their view "this is to prepare the management for acceptance of software and to get them to participate in the development effort" (Ramingwog et al. 2017, p. 344).

Stage II: Requirements Elicitation. Two stages of Faat, Preparation of Product Backlog and Evaluation were put in this stage. Evaluation in this case pertains to assessment of developer performance at the end of each stage. The other stages included are part of the Planning stage from PXP1 (activities here are eliciting user requirements and formulation of system metaphors), Requirements stages from PXP2 and Scrum – Solo and part of the activities from the Kick – off –Meeting and User story from Go – Scrum (the meeting activity). The Planning and analysis stage of DeSoftIn also fits into this stage. Since DeSoftIn is designed for an academic environment, an important practice at this point is the defining of a project and product scope. Project scope refers to all the work to be undertaken in the project, while product scope captures the functionality to be delivered by the product. While the scope is set here, the method recommends its adjustment as per need as the project progresses.

Emerging theories:

i. *The use of a prioritised product backlog helps to keep track of project progress and promotes product completeness*

In Bernabé, Navia and García-Peñalvo (2015, p.689), a product backlog is described as a tool to capture and prioritise all tasks, keeping track of the executed and outstanding tasks. González-Sanabria, Morente-Molinera and Castro-Romero (2017) recommend the use of a checklist at this stage that links user requirements to user roles. Such a checklist enables the developer to have full control over the development process as they know which user to consult at each stage.

ii. *Simple metaphors encourage product understandability and testability.*

Metaphors are used to describe the system from the user's perspective. Thus, if used for system representation should facilitate understanding of the requirements (Agarwal & Umphress 2008, p.84) by both the developer and the users.

iii. *Task automation facilitates product reusability and timely product delivery.*

Identified repeating tasks should be automated to allow for future use (Bernabé, Navia & García-Peñalvo 2015, p.694). To deliver timely projects, the developer needs to automate most of their work (Dzhurov, Krasteva & Ilieva 2009, p.253). Automation reduces developer effort as it minimises rework associated with human error. All in all, developer productivity is enhanced through automating recurring tasks.

Stage III: Release and Sprint Planning

Most of the activities in the methods analysed have been grouped into this stage. The stage includes Creation of User stories, Iteration Initiation and Planning from Faat (Bernabé, Navia & García-Peñalvo 2015), Planning from PXP1 (Agarwal & Umphress 2008) and PXP2 (Dzhurov, Krasteva & Ilieva 2009), Project Planning from Go-Scrum (Ramingwong, Ramingwong & Kusalaporn 2017) and Management from Solo Scrum (Pagotto *et al.* 2016). Part of Planning and analysis from DeSoftIn also falls into this stage. Most authors concur on the creation of user stories to capture user requirements. User stories capture user requirements in a simple and easy to use way.

Bernabé, Navia and García-Peñalv (2015, p. 688) recommend the use of the acronym INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable) to ensure simplicity of user stories. INVEST is an acronym popularised by most agile methods (Heck & Zaidman 2018,

p.143). Using this approach, user stories should be independent of each other to facilitate the delivery of the product in components. They should be designed to be negotiable, so that at any time the concerned stakeholders can request for changes in the deliverable associated with the user story without affecting any components already running at the user's site. All user stories should add value to the system under development. Similarly, user stories should be small enough to facilitate accurate resource and time estimation. User story testability is an important part of iterative development. Each user story should enable the development team to write acceptance tests used to test the software component associated with the user story at iteration end. This importance of simplicity in user stories is supported by Ramingwog, Ramingwog and Kusalaporn (2017, p. 345) and by Agarwal and Umphress (2008, p. 84) who recommend the use of a metaphor simple enough to facilitate system understandability.

In González-Sanabria, Morente-Molinera & Castro-Romero (2017) a recommendation to plan for risk management is given. The developer is encouraged to identify all those activities that might pose risk to the quality of the software product or the time of project completion. A risk management plan should be created indicating risk owners for each identified risk. This enables the developer to quickly consult those concerned in the event that the risk materialises. From the activities organised into this stage the following theories emerge:

i. *Small user stories promote product simplicity*.

In creating user stories: "...clarify everything the product will offer, to list all the operations that users can perform,…., must be divided in smaller, simpler, achievable and estimable user stories" (Bernabé, Navia & García-Peñalvo 2015, p.692).

ii. *Product refactoring and use of simple story cards result in product simplicity* (Bernabé, Navia & García-Peñalvo 2015; Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009).

iii. *Use of a work breakdown structure (WBS) in planning promotes product completeness* (Dzhurov, Krasteva & Ilieva 2009; Pagotto *et al.* 2016)

iv. Size and time estimation in planning reduces schedule slippage (Bernabé, Navia & García-Peñalvo 2015; Agarwal & Umphress 2008).

Dzhurov, Krasteva and Ilieva (2009, p. 254) indicate that for first time projects, size and effort estimation suffers from in-availability of data to base estimates, and might not produce

expected results. The developer is therefore recommended to review estimates at the end of iterations to reflect the knowledge acquired during the development process.

v. *Small milestones and releases encourage timely delivery* (González-Sanabria, Morente-Molinera & Castro-Romero 2017,p.28; Bernabé, Navia & García-Peñalvo 2015,p.689) .

Milestones and releases mark project progress. Developers using Faat should adhere to the following advice; "milestones and releases should be maintained small enough to keep things in perspective and not to take the risk of employing a lot of time on features that may not be delivered on time" (Bernabé, Navia & García-Peñalvo 2015, p.689). In González-Sanabria, Morente-Molinera & Castro-Romero (2017, p.28), the developer is advised to use release iterations of three to ten days. This visualises the development process and helps to keep the user informed about development progress. These theories form a guideline on the practices in the Requirements and Elicitation stage.

Stage IV: Development with Review

At the development stage, the code for the software product is written. To enable the delivery of quality code, most reviewed authors recommend constant review of one's code before integrating with the baseline code. The following stages from the studies reviewed have been grouped to give the Development with Review stage:

Development and Review stages from Faat (Bernabé, Navia & García-Peñalvo 2015), Development from PXP1 (Agarwal & Umphress 2008), Design and Implementation stages from PXP2 (Dzhurov, Krasteva & Ilieva 2009), Release and Sprint Planning stage and Release with Inspection stage from Go – Scrum (Ramingwong, Ramingwong & Kusalaporn 2017) and the Sprint and Management stage from Scrum Solo (Pagatto et al. 2016). The Design and Development stages of DeSoftIn also fall under this stage. An analysis of activities in this stage gives the following themes:

i. *Use of version control enhances product maintainability* (Bernabé, Navia & García-Peñalvo 2015, p.690).

ii. *Test driven development and unit testing enhances code quality* (Dzhurov, Krasteva & Ilieva 2009, p. 258 ; Bernabé, Navia & García-Peñalvo 2015, p. 690).

iii. *Refactoring enhances system extensibility and maintainability* (Dzhurov, Krasteva & Ilieva 2009, p. 258 ; Bernabé, Navia & García-Peñalvo 2015, p. 690).

iv. *Prioritisation of tasks during development enhances user acceptance* (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.27).

v. *Time estimation review improves future estimates and reduces development bottle necks* (Dzhurov, Krasteva & Ilieva 2009¸p. 256; Bernabé, Navia & García-Peñalvo 2015, p. 693).

vi. *Frequent customer communication reduces required documentation* (Agarwal & Umphress 2008, p. 85).

vii. *Use of a dummy programming partner and objective self-criticism improves code quality* (Bernabé, Navia & García-Peñalvo 2015, p.691; González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.27).

Most of the theories emerging from this stage are well established in software engineering. Unique to solo software development is that explaining program code to a dummy object facilitates the discovery of errors in the code. (Bernabé, Navia & García-Peñalvo 2015, p. 691). The recommendation is that as one explains one's code to the dummy, one is likely to uncover errors in one's code. This concept is corroborated by González-Sanabria, Morente-Molinera and Castro-Romero (2017), who recommend that the developer objectively practices self-criticism on all development practices. If done carefully this is likely to improve the quality of the delivered products.

Stage V. Sprint Review and Close

A sprint is designed to deliver functionality at the user' site. At the end of each sprint the delivered component should be assessed for compliance with the requirements. The following stages from the component methodologies have been included; Iteration and Evaluation from Faat, System testing from PXP2 (Dzhurov, Krasteva & Ilieva 2009), Management and Deployment stages from Scrum Solo and Sprint from Go-Scrum. The following theories can be derived from this stage:

i. *Consistent sprint reviews encourage customer communication* (Ramingwong, Ramingwong & Kusalaporn 2017, p.3467).

ii. *Early fixing of errors enhances product quality* (Dzhurov, Krasteva & Ilieva 2009, p.256).

iv. *Performing of acceptance tests promotes product correctness* (Bernabé, Navia & García-Peñalvo 2015, p. 690).

Stage VI. Evaluation

The last stage of the framework drawn from the studies is Evaluation. This consists of two stages with the same name of Evaluation, drawn from Faat and DeSoftIn, Post Mortem from PXP1 and Retrospective from PXP2. Evaluation performed during the development process helps to improve developer productivity as well as refocus the development process. If performed at the end of the project, it serves as a knowledge creation process for improvement in future projects. Since the developer performs most of the development activities single handed, they are encouraged to involve the customer in the evaluation process. DeSoftIn recommends the involvement of a consultant (or supervisor) who assists the developer to discover new ways of improving the development process.

Some activities from Go - Scrum and Scrum Solo are pushed down to this stage. These include the Sprint Review meeting of Go – Scrum (Ramingwog et al. 2017, p. 345) and the Validation activity of Scrum Solo (Pagotto *et al.* 2016)Validation is an important concept in software development. It serves to confirm that the developer has built the right product for the customer. From these activities, minimal data can be derived. The following theories are deduced:

i. *Correction of methodology practices early in the development cycle minimises project failure* (Dzhurov, Krasteva & Ilieva 2009, p. 256)

ii. *Product validation before final deployment ensures software meets user requirements* (Pagotto *et al.* 2016, p.6; González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.27).

The synthesis of the concepts from the participating studies helps this research to derive a quality theory for the resulting framework. A theory in this case is considered as a set of relationships about constructs in a field of study (Gregor 2006, p.615). The derived relationships can be expressed in the form of a conceptual model (Mohammed et al. 2016, p.698) as shown in Figure 2.4 and Figure 2.5. Two broad theories emerge, the product and general software development theories. Figure 2.4 shows the product quality theories, while Figure 2.5 shows general software development theories.

Figure 2.4 shows that the adoption of development standards, use of small user stories and tasks, automating code reviews, writing testable code and refactoring promote simplicity and thus quality of product. At the same time the use of development standards and product validation promote product consistency. Similarly, use of simple metaphors to capture user requirements promote product understandability and code quality. The rest of the figure is interpreted similarly.

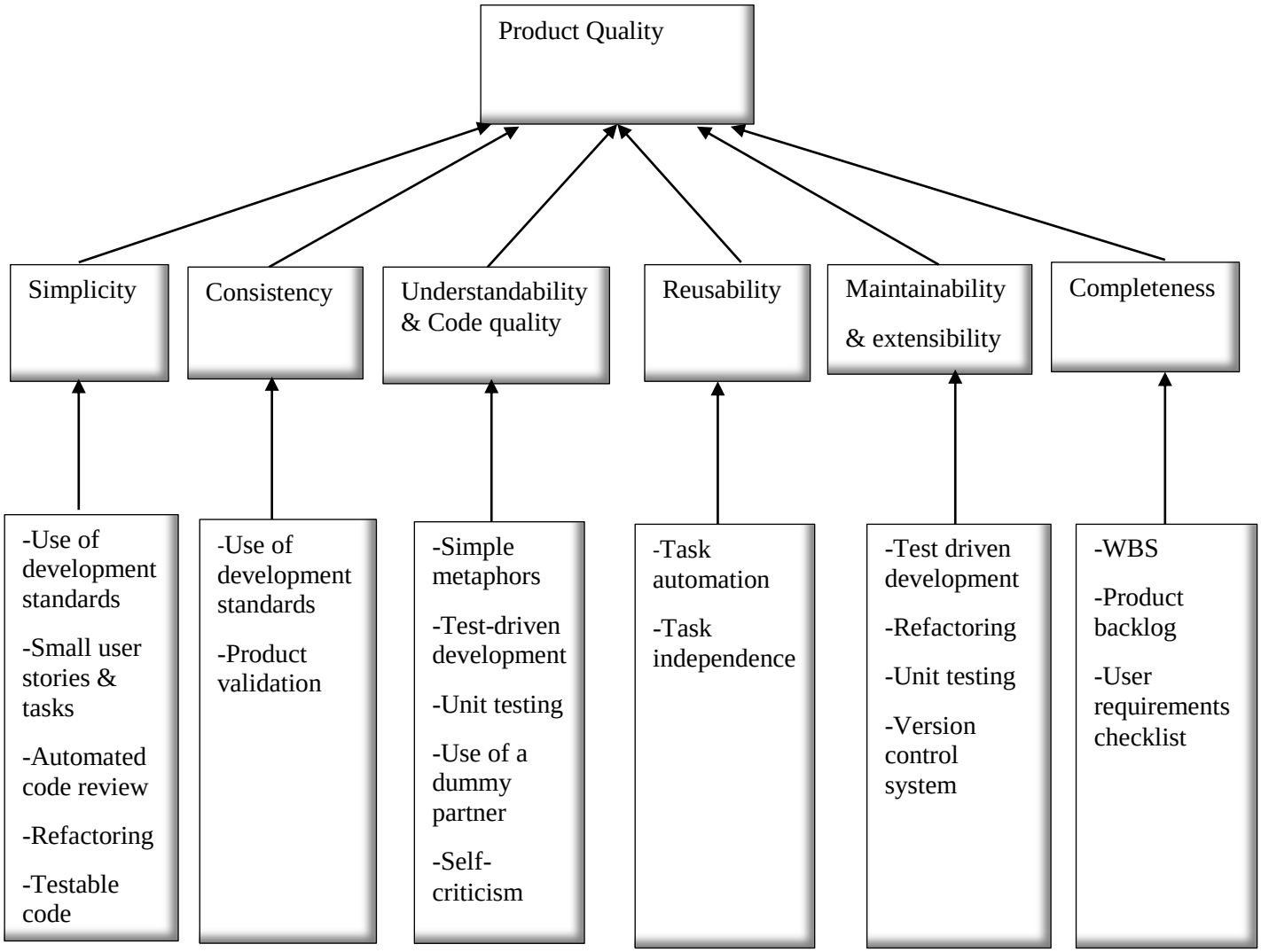


Figure 2.4: Product quality theory

Some general software quality theories describing the development process were also observed from this synthesis. These practices do not directly impact the quality of the product, but contribute to the success of the development effort. Figure 2.5 shows general theories

derived from the studies synthesised. From the figure it can be seen that user education on methodology, early user involvement and task prioritisation promote user acceptance.

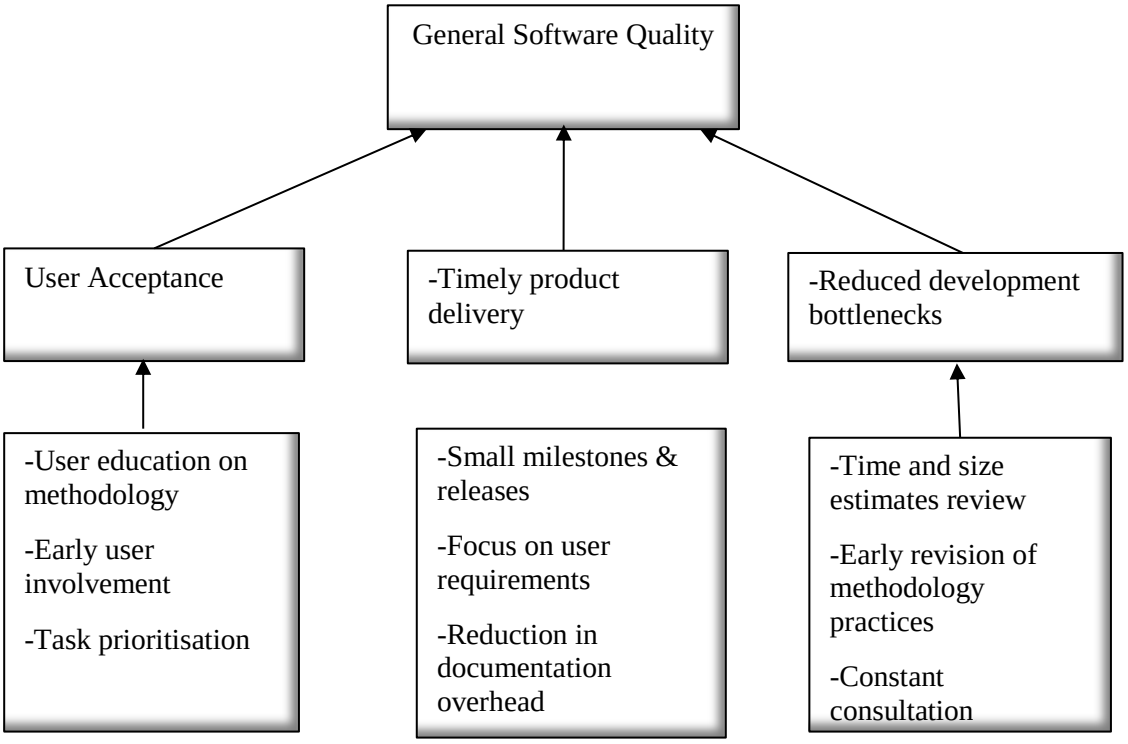


Figure 2.5: General software quality theory

Step 7: Reporting the Study

This meta-ethnography has resulted in the formulation of two broad theories regarding the development of high-quality software products in a solo development environment. The product quality theory stipulates that simplicity, consistency, understandability, reusability, maintainability and completeness promote high product quality. On the other hand, from a general software development process, user acceptance, timely product delivery and reduced development bottlenecks promote the general software development process resulting in high-quality software. The generated theories form a guide for methodology designers and provide a basis for the formulation of a high-quality methodology, which is the main reason for conducting this review and carrying out this research.

2.5.4 Threats to validity

According to Runeson et al. (2012, pp. 70-72) the validity of a study determines the acceptability of its results by the target community. From these authors' perspective, four kinds of validity need consideration in a qualitative study like this. These are construct, internal, external, and reliability (p.71). Construct validity refers to the dependability of the structuring of the study to answer the posed research questions. This means the study setup should be such that, results obtained using the setting provide unbiased answers to the study questions. Internal validity relates to the planned handling of unexpected interactions of variables in causal relationships, which may falsify the findings of a study. Researchers should make all the effort to identify such variables and plan to counter their influence on the results. External validity pertains to the generalizability of the results of the study to other populations outside the study. Reliability pertains to repeatability of the study by other researchers to get similar results. The next paragraphs discuss how these four forms of validity were addressed in this meta-ethnography.

In addressing the issue of **construct validity**, research questions on the meta-ethnography were formulated to be confined to the SSDM environment. Only articles by first author discussing the methodology were retrieved from research outlets publicising software engineering research. The research restricted the articles to only those discussing quality practices in SSDMs. To minimise missing some articles, the researcher also used Google Scholar to search for solo software development publications. With all the efforts made, some articles may not have been published in the outlets mentioned so far. To address this threat, the researcher checked the references of the articles found using database searches to identify any such sources. To ensure quality in the synthesis, the inclusion and exclusion criteria set at the study onset were reviewed by the academic supervisor for consistency and coverage of the articles of interest.

To deal with the **internal validity** threat, the researcher iteratively went through the stages of the meta-synthesis, referring to the original data at every stage, and including quotes directly from the source data to capture the concepts in the studies involved. In deriving the theory, a systematic approach to compare and contrast concepts in the studies was adopted. The resulting abstractions from the synthesis were submitted to the PhD supervisor for further scrutiny so as to deal with bias due to the researcher's interest. Further, as the quality of the abstractions are dependent on the quality of the accounts included in the synthesis, the

researcher only used peer reviewed studies from recommended scholarly sources as primary data sources (Sandelowski et al. 1997, p.368).

External validity in this case pertains to the generalisability of the synthesis results to all solo development environments. The participating studies in the review are drawn from different backgrounds, ranging from business, to academic. Since the research has included studies discussing methodologies from varied environments, the quality theory results of this study can be generalised to any solo development environment. While the quality theory pertains to solo developers, this research does not rule out the applicability of the quality concepts cited in this thesis to team environments. The theory's applicability to teams needs proof through empirical studies.

In a meta-ethnography, the main aim is to derive higher levels of data abstraction, based on all the available primary studies. Researcher bias may impact on the quality of the abstractions produced. To minimise researcher bias, transparency in data collection and analysis is encouraged. In a bid to ensure **reliability** in this synthesis, guidelines from Noblit and Hare (1998) supported by suggestions from Sandelowski et al. (1997) were used to perform the meta-ethnography. As suggested by the latter, a template generated from the data was used to extract and analyse the concepts of interest. The researcher made all efforts to support all extracted concepts with quotes from the source data. The quality framework generated by the meta-synthesis was subjected to peer review at a research seminar and presented at an international conference (Moyo & Mnkandla 2019). Feedback obtained from the participants in these cases was used to improve the quality theory generated from the synthesis.

While the measures above were put in place to deal with the threats to validity, there are limitations in this study. One of the limitations is that non-electronic studies or those studies published in databases not included in the study might have been missed. Further, the study did not include those studies that were not formulated as methodologies. This would mean quality practices in such studies were not included in the study. The data in this study is therefore representative of only those studies participating in the meta-ethnography. The other limitation is that some quality theories may not have been captured in this study due to researcher bias, although efforts were made to subject the theory generation process to different audiences.

2.6 Exposing the gap in SSDMs

The aim of this research was to design a solo software development methodology that embeds practices which promote quality in the developed software product. Since quality is a complex phenomenon, the product quality characteristics as defined in the ISO/ IEC 25010 model were used as a benchmark against which to measure the software product quality. The model and the reason for opting for it were discussed in Section 2.4.1. The model defines abstract quality characteristics which are: compatibility, functional suitability, maintainability, performance efficiency, portability, reliability, security and usability. The abstract characteristics in turn are described by measurable concrete characteristics as shown in Figure 2.3. A mapping of the quality theory generated by the meta-synthesis carried out in this thesis against the model shows that some concepts of the theory appear at the abstract level while others appear at the concrete level. This mapping is shown in Figure 2.6. The abstract characteristics that can be mapped directly are maintainability and functional suitability (completeness). The other characteristics such as usability, portability and reliability are supported by sub – characteristics at the concrete level. It was noted that of the abstract characteristics supported at the concrete levels, none is fully supported.

The mapping also shows that there are some abstract characteristics in this model that are not supported. These are security, compatibility and performance efficiency. This mapping has therefore exposed a gap in the existing methodologies regarding the promotion of quality products as defined by the ISO/ IEC 25010 model. In progressing knowledge in the SSDM, this research therefore sought to identify security promoting practices from existing lightweight methodologies that are compatible with the existing quality practices. When integrated with the quality practices in the derived framework, it was hoped that these would build security into the developed software.

Software projects tend to be different in nature, due to varying team sizes, different environments, different budgets and time frames (Pardo et al. 2011, p.94; Hughes & Cotterrell 2012, pp.61 - 67). This research focused on a team size of one. The uniqueness of solo software development environment was discussed in Section 2.5. One unique feature of the solo development environment is the limited resources, which impacts on budgets for training (Coleman & O'Connor 2008, p.773; Basri & O'Connor 2010, p.1457).

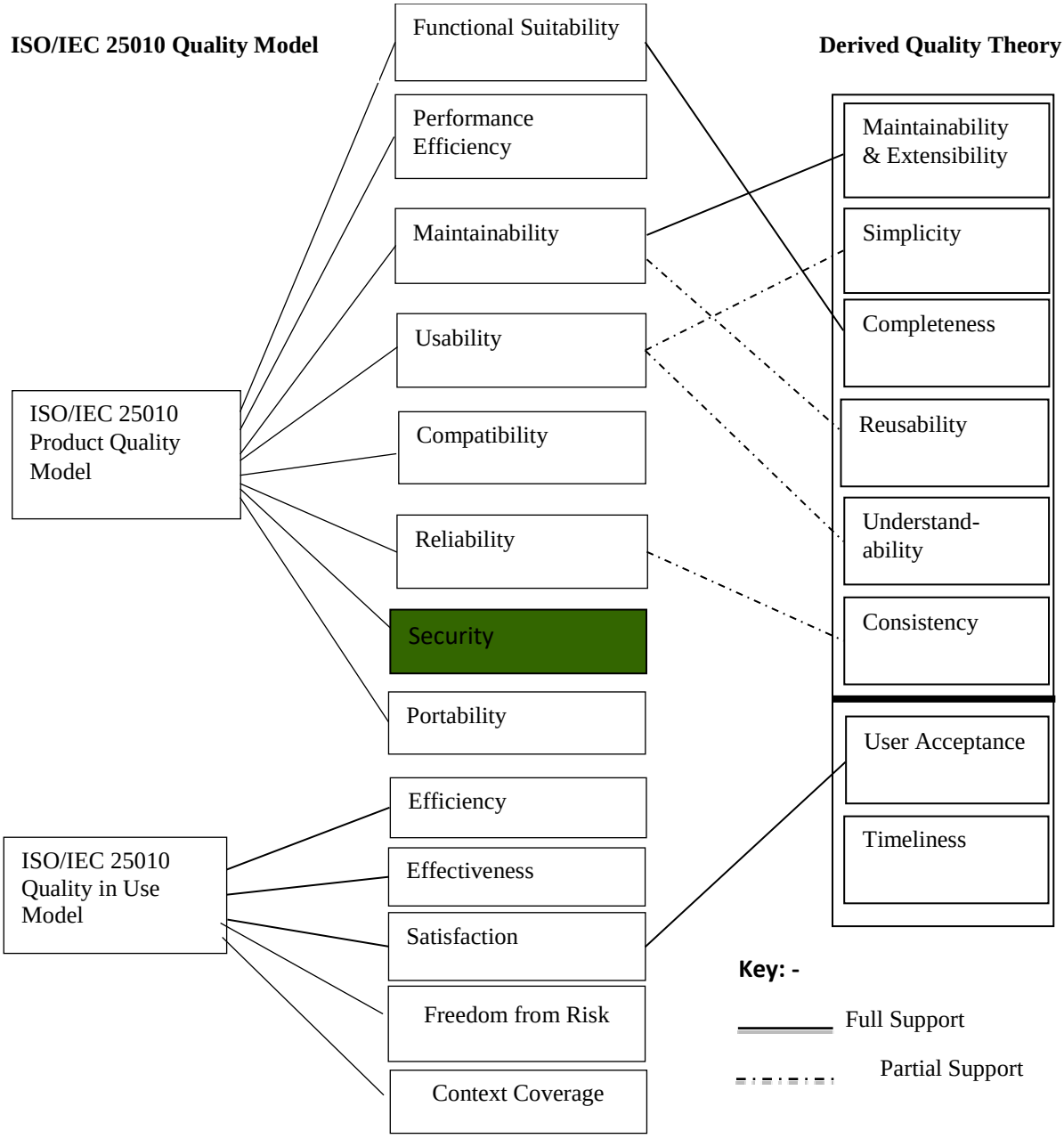


Figure 2.6 : Mapping quality practices to ISO/IEC 25010 quality model

An ideal methodology for such an environment should therefore be adaptable, so that it can be used in a number of projects with minimal adjustments. To achieve such flexibility, this research proposes the definition of a method core (Keramati & Mirian-Hosseinabadi 2008, p. 751) that can easily be extended depending on the type of software product under development.

2.7 Tools for Methodology Design

The success of methodology design and implementation is heavily dependent on the tools used for the purpose. A search of the literature reveals two popular frameworks for method engineering. These are the Eclipse Process Framework (EPF) Composer and the Essence Framework (Elvesæter et al. 2013, p. 1). The EPF Composer is the most preferred in software engineering methodology creation as it is an open framework, has a number of plug ins and supports several modelling and programming languages. Further, the framework supports methodology flexibility and extensibility, as it enables the definition of activities and tasks that are independent of each other. The latter promotes task reusability (Porres et al. 2013, p.269). This is a favourable property of the framework for the methodology proposed in this thesis as it addresses the issue of resource scarcity. Other researchers have also used the framework in designing similar products. Elvesæter, Benguria and Ilieva (2013, p.1) used the framework to develop and implement the agile REMICS methodology. Mtsweni (2013, p. 122) used EPF to design a framework for developing intelligent semantic services. This research adopts the EPF composer as the main platform for method engineering as it is an open platform, thus is readily available and can be used in the development of lightweight methodologies. The use of the EPF Composer in implementing the Secure-SSDM is discussed in Section 5.3

2.8 Chapter Summary

The literature review conducted in this chapter has provided this research with a background and theory upon which to base the development of the proposed Secure-SSDM. The overview of the software development landscape in general, and the in-depth analysis of the solo software development environment in particular, provides a rich base for building the proposed methodology. Having thoroughly analysed existing SSDMs, it has been possible to explicitly show the gap that still exists in the solo software development environment. The meta-synthesis performed on the former enabled this research to position this study in line with what still needs to be done in order to progress knowledge in the field.

In Chapter 3, the approach used to develop the Secure-SSDM is deliberated on. The careful setting of the methodology development is meant to promote the success of the project. Careful formulation of the research roadmap also enables other researchers to give respect to the resulting methodology at the same time enabling repeatability of the process.

CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction

In Chapter 2, a review of the literature on the current landscape in small-scale and solo software development was conducted. An in-depth literature review of existing SSDMs was performed resulting in the preliminary design of an SSDM framework synthesised from these. Preliminary quality theories were also derived from existing quality practices. A comparison of the formulated quality theories with the ISO/IEC 25010 software quality model (ISO 2010) revealed that, although the derived quality theories supported quality characteristics defined in this model, there were no practices to support security, compatibility and performance efficiency.

The absence of security practices in the reviewed SSDMs is consistent with the observation by Mohammad, Alqatawna and Abushariah (2017, p.814). In their study, the authors conclude that many software development methods do not support security in their phases of the software development life cycle (SDLC). This is not surprising for lightweight methods, as focus on improving software security is viewed as reducing productivity and increasing costs (Baca & Carlsson 2011; Mohammad et al. 2017, p.817). Further, agile methods on which this research focuses, have been shown to lack security promoting practices by a number of researchers (Aguda 2016, p.6; Karim et al. 2016, p. 5334; Rafi et al. 2015, p.380; Wäyrynen et al. 2004, p.127). Although security was not the only characteristic missing in the theory generated by the literature review, in this Internet age where most applications are deployed on the World Wide Web, the need to address the security issue of the derived framework is compelling.

Besides revealing the quality gaps in existing SSDMs, the literature review further helped in shaping this research as it clearly provided a direction of what questions to ask. Guided by the literature review this research confidently poses the following question and sub-questions:

How can a secure-SSDM be developed to enable quality and security in the developed software?

The following sub questions were further posed to help answer the main question:

1) *What methodologies exist for solo software development?*

2) *What practices and techniques in the existing methodologies promote quality in the developed software?*

3) *What lightweight practices and techniques in the software development life cycle support software security?*

4) *How can the identified practices and techniques be integrated into a Secure-SSDM to enable quality in the final software product?*

5) *How can the resulting methodology be evaluated?*

The design and implementation of a methodology that incorporates security promoting practices is viewed as one of the main contributions to knowledge of this research. This is the reason for posing a separate question on security as a quality characteristic. This research concurs with Al-amin et al. (2018, p.33) that incorporating security practices into the software development process promotes security in the resulting product. However, integrating lightweight quality and security practices is not an easy task (Ragunath et al. 2010; Rindell et al. 2018; Sonia et al. 2014; Sonia & Singhal 2012; Keramati & Mirian-Hosseinabadi 2008). There is need therefore, to develop a systematic approach for the purpose. To that effect it was necessary to identify and adapt an established method to guide the integration process. Searching the literature enabled this research to identify practices integration algorithm by Keramati and Mirian-Hosseinabadi (2008). This was then adapted for the purpose. Using the algorithm, lightweight security promoting practices were identified from existing secure software development methods and incorporated into the framework derived in Chapter 2 to build a novel high-quality Secure-SSDM.

Since the literature review had conclusively shown that existing SSDMs lack security promoting practices, lightweight secure software development processes provided an alternative source for identifying those security promoting practices that could be undertaken by a single developer. The following sections discuss the research paradigm, the research method and the tools used to design and implement the proposed Secure-SSDM.

3.2 Research Paradigm

A philosophical research paradigm is one's perception about the world around them and how one builds on those perceptions to create knowledge (Oates 2006, p. 282). A number of research paradigms exist. Each paradigm is distinguished by the following dimensions:

ontology, epistemology, axiology and methodology. These paradigmatic dimensions influence how one conducts and constructs knowledge from research. This section overviews the main research paradigms showing their dimensions of ontology, epistemology methodology and axiology. Exploring these is important for the purposes of identifying a suitable paradigm in which to pursue this research. Taking a stance in which to approach research helps to give credibility to the research, at the same time assisting stakeholders in evaluating the quality of that research.

First, the dimensions used in differentiating the paradigms are outlined. The ontology of a paradigm refers to the nature of reality in that paradigm. Reality can either be concrete or abstract (Vaishnavi et al. 2017, p. 24). This means, reality can be dissected into what is tangible and what is intangible. Ontology therefore, refers to one's perception of reality around them (Wahyuni 2012, p. 69). Researchers in the various paradigms perceive reality differently.

Epistemology on the other hand refers to the acceptable and effective ways in which knowledge is generated and used in a paradigm. This dimension defines knowledge dependencies and means of affirming the existence of knowledge. It defines the nature of knowledge in a given paradigm. To be credible, research in a given paradigm should be conducted according to what is accepted as the norm in that paradigm.

Axiology considers the acceptable roles a researcher can play in a researched environment. It defines what is ethically acceptable, and that which is not. Vaishnavi et al. (2017, p.24) refers to axiology as the values held by a researcher and the reasons for holding those values. The axiology of a researcher determines the acceptable associations among what is researched and the researcher.

Methodology as a dimension defines the approach of conducting research in a particular paradigm. It provides a model for carrying out the research. The methodology standardises the research process, enabling repeatability of research. These four dimensions distinguish existing research paradigms and need careful consideration in any research. In deciding what paradigm to adopt in conducting research, care should be taken to consider these dimensions, and choose a befitting paradigm.

The four paradigms applicable in this thesis that need consideration before settling for an appropriate one(s) are: positivism (reductionism), interpretivism (constructivism), critical research and design science research (Hevner et al. 2004; Vaishnavi et al. 2017; Oates 2006;

Easterbrook et al. 2008). These differ based on their ontology, epistemology, axiology and methodology. In the following paragraphs these differences are considered with the intention of settling for a befitting paradigm (s) for this research.

Positivism axiologically upholds that concepts in the world exist independent of researchers. This means that these concepts can be studied objectively without the researcher's interference (Oates 2006 p. 286-287). The epistemology of positivists is that knowledge is created through logical inference of observable facts about the concepts and their surrounding world. In creating knowledge, large concepts are usually broken down into smaller ones, so that if a fact is proved to hold in the small isolated components, then it also holds in the larger concept (Easterbrook et al. 2008, p.291). In this paradigm the objects of study are removed from their original setting and studied in an artificial environment. This approach was deemed inappropriate for studying a software development methodology whose success is heavily dependent on the environment of application and the people using the methodology. What makes the positivist approach inappropriate for this study is its dissociation of the object under study from its environment, making it unsuitable for studying socio-technical artefacts like software development methodologies and associated software products.

Interpretivists on the other hand create knowledge through meanings derived from observing concepts in their surroundings. They formulate theories based on the meanings of what they observe around them at that moment in time. Knowledge creation in this paradigm depends on the researcher's understanding of the environment. This knowledge is also time dependent. From a computing perspective, Interpretivists study the way humans create computer systems, how they are influenced by and how they influence these systems (Oates 2006, p.292). Interpretivism presents a viable option for this research as it supports the design of a methodology for a specific set of developers to address their needs in a specific setting. This paradigm was deemed ideal for creating knowledge from the existing SSDMs, and for deriving the developer's perceptions of the utility of the Secure-SSDM. It enabled the studying of software development and the associated SSDMs as social practices heavily influenced by developers (Dittrich 2016, p.751).

A similar paradigm to interpretivism is critical research. Critical research like Interpretivism subscribes to the influence of human perception in knowledge creation, but further seeks to understand the systems that influence the creation of that knowledge (Oates 2006, p,296). Critical researchers seek to bring balance into unbalanced situations by suggesting means of

empowering the disadvantaged in the research environment. The open source movement is an example of critical research. Open source proponents aim at availing computing solutions to the economically challenged (Easterbrook et al. 2008, p.292). This approach was rendered inappropriate in this research as there are no situations of imbalances to be addressed by this research.

The fourth paradigm of interest is Design Science Research (DSR). DSR as a paradigm, acknowledges the existence of several world states. It focuses on creating innovative artefacts, and evaluating these artefacts' capabilities to move the world between these states (Hevner et al. 2004, p.98; Vaishnavi et al. 2017, p.25). Epistemologically, researchers in this paradigm build knowledge by designing and introducing novel artefacts into the world from which they create new knowledge through circumscription. A researcher in this paradigm iteratively introduces modified artefacts to an environment to bring about change to that environment. Knowledge here is created by observing the artefact's interactions with the environment. The predictability of the artefact's behaviour when introduced to an environment defines truth in this paradigm.

Table 3.1 summarises the research perspectives discussed in the preceding paragraphs. It gives a comparison of the four paradigms considered in this research in terms of the dimensions explained above. As shown in the table, positivists believe in a single knowable reality, while interpretivists subscribe to multiple realities, which are dependent on the environment. This multiple-realities perspective is shared by critical realists, who further acknowledge the influence of both the environment and external sources on these realities. Similarly, design science researchers subscribe to multiple realities which are associated with different world states. Such realities are brought about as artefacts are introduced to an environment, to move realities from one state to the other. The best reality is that which achieves the expected results in a given environment.

Using Table 3.1, and considering the problem at hand, this research adopts DSR as the main paradigm. DSR is viewed as the best option, as it facilitates the building of an artefact that can be iteratively refined until satisficing utility is obtained. At each iteration, as the artefact is introduced to the environment, it is evaluated and refined until it exhibits the desired characteristics that address the unique needs of solo developers. During the process of refining the artefact, there is need to understand the utility of the artefact from the developers' perspective. For this purpose, the interpretivist paradigm was adopted as a complementary

paradigm to enable the understanding of the perceptions of the developers on the utility of the Secure-SSDM. The Interpretivist paradigm also influenced the formulation of the primary framework for the artefact. At that stage, the researcher abstracted meaning of the quality practices as perceived by the authors of the SSDMs participating in the meta-synthesis performed in Chapter 2.

Table 3.1: Research paradigms and their dimensions: adapted from Wahyuni (2012, p.25); Vaishnavi, Kuechler and Petter (2017, p.25)

Dimension	**Research Perspective**			
	Positivist	**Interpretive**	**Critical Research**	**Design Science Research**
Ontology	Single knowable reality; probabilistic	Several realities, socially constructed	Several realities, constructed by the environment and external sources	Multiple, contextually situated alternative world- states. Socio-technologically enabled
Epistemology	Objective, dispassionate researcher detached from the environment	Subjective, researcher-participant dependent	Dependent on what can be seen, Knowledge created from concepts and their contexts	Knowing through making: objectively constrained construction within a context. Iterative circumscription reveals meaning
Methodology	Observation; quantitative, statistical	Participation; qualitative. Hermeneutical, dialectical	Can use both the quantitative and qualitative forms	Developmental. Measure the artefact's impacts on the system, Uses mixed methods
Axiology	Truth: universal and beautiful; prediction	Understanding: situated and description design	Researcher background influences research outcomes	Control & creation; Researcher values impacts on outcomes

3.3 DSR Research Methodology (DSRM)

A research methodology provides an architectural guide to research in a given philosophical paradigm (Wahyuni 2012, p. 72). A methodology's purpose is to structure the study by laying down the steps, activities and tools to be used for the research as well as providing means for evaluating that research. Since DSR is the overarching paradigm in this thesis, there is need to adopt a methodology in line with this paradigm.

Some researchers have proposed guidelines for conducting DSR. Hevner et al. (2004) emphasize on a DSR methodology that promotes: the design of a valuable artefact to a given audience; relevancy of the artefact to the problem being solved; rigour in the design of the artefact; rigour in the evaluation of the utility of the artefact in the environment for which it is designed; the utilisation of existing knowledge to build new knowledge in the field; and the presentation of both the artefact and new knowledge generated to relevant stakeholders. Taking a cue from these authors' guidelines, Peffers et al. (2008) suggest a design Science research methodology (DSRM) for undertaking research in this paradigm. Their methodology provides a systematic approach to designing and evaluating the utility of the artefact under design. Apart from providing the researcher (s) with an organised evaluation approach, it guides research reviewers in judging the quality of a DSR endeavour. Peffers et al. (2008, pp.52-56) DSRM can be summarised using the following steps:

1.Identifying the problem – At this stage, the researcher identifies the problem (or opportunity) through discussions with people, observation of the world around them or review of various forms of literature. At this stage, the significance of the solution is also identified as it gives reason for pursuing the research.

2. Defining solution objectives – Based on the problem, the researcher proposes objectives to be addressed by the solution in order to solve the defined problem. These solution objectives are used to evaluate the artefact at the end of the study. Objectives determine the quality of the artefact. They can either be formulated to be quantitative or qualitative showing how the artefact will solve the identified problem. The objectives of the Secure-SSDM are detailed in Chapter 4.

3. Designing and developing the proposed artefact – The artefact is produced at this stage. Appropriate activities, tasks and rules are adopted to design, implement and document the artefact. This is heavily dependent on the artefact to be produced. Activities carried out to

develop a model would be different from those executed to design a method. The artefact in this study is a software development methodology. Chapter 5 discusses the methodology used to design the Secure-SSDM.

4. Demonstrating the utility of the artefact –As suggested by Peffers et al. (2008, p.55), this entails using the artefact to solve a representative problem in the area. Demonstrating the utility of a software development methodology entails using the methodology to design and develop quality and secure software products. This follows from the solution objectives. In this thesis, a multiple case study was used for the purpose.

5. Evaluation – Evaluation measures the utility of the artefact based on its performance from the demonstration stage. Various forms of measures can be used. Examples include qualitative evaluation of the target audience perception of the utility of the artefact. Other forms of evaluation include quantitative measures of the artefact's performance, use of simulations, or the use of satisfaction surveys (Peffers *et al.* 2008, p.56). Results obtained from the evaluation process are used to determine whether to refine or release the artefact for use.

6. Communicating the results of DSR research – This involves the use of appropriate channels to publicise the artefact, its design process, its evaluation process and the outcome of the evaluation. Channels such as academic conferences, journals, book chapters or magazines may be used for the purpose.

The following subsections detail how this DSRM was used to build the Secure-SSDM. In using the methodology, suggestions by Vaishnavi, Kuechler and Petter (2017, p.11) to generate new knowledge during design were utilised. While these authors' original knowledge generation cycle is based on Hevner et al. (2004)'s five stage DSR process, the similarities in the two processes were used in this thesis to generate the knowledge flows. Figure 3.1 summarises DSRM steps and associated knowledge generation processes. As shown by the circumscription and the SSDM knowledge arrows, DSRM is an iterative process. Circumscription refers to the discovery of new knowledge when things do not work as expected for the artefact under development, forcing the researcher to dig deeper into existing knowledge in order to make the artefact work (Vaishnavi, Kuechler & Petter 2017). Just like the design process, circumscription is an iterative process that generates new knowledge during the iterations. Circumscription together with "abstraction and reflection" at the end of the research help to contribute knowledge to the existing SSDM knowledge base, which is the distinguishing feature of DSR.

In this research, the knowledge base contains SSDM practices, their relationship with quality characteristics and development processes. Besides the artefact being the main contribution in this research, knowledge contribution is another important contribution of this thesis. This was achieved at various points of the design cycle. Knowledge contributions from this thesis are summarised in Section 7.3.

As Figure 3.1 shows, knowledge generation starts during the design and development of the artefact. As the researcher discovers that some processes obtained from the knowledge base do not work, the latter is updated with those processes that do. The resulting artefact is then demonstrated through application in an appropriate environment. Similarly, any new discoveries from this application are added to the base. During the demonstration process, the artefact is evaluated against the originally set objectives. Results from the evaluation process are used to update the knowledge base. Once the researcher is satisfied that the artefact meets its intended objective, the research stops. The new artefact may further stimulate new research based on its performance in its intended area. The next section elaborates on the application of DSR in this thesis.

3.3.1 Identifying the problem

The researcher's academic background in software engineering stimulated interest in the area. Having observed students adapt methodologies for use in their final year individual software development projects inspired this research. Reviewing the literature over the years in search of an appropriate method to guide students showed that minimal research exists in this area. A systematic literature review on solo software development (SSD), conducted in this thesis further proved that previous studies have ignored the SSD environment. The small number of studies (seven in this case) published in mainstream software engineering outlets confirmed this.

Apart from the small number of studies found, the synthesis of the retrieved SSDMs further showed that existing methodologies' support for the development of quality software products is limited. Security as a quality characteristic is not supported by existing SSDMs. The research efforts to improve SSDMs have not necessarily translated to improving quality of the SSDMs, particularly in terms of secure software development. Section 2.5 demonstrated that existing SSDMs have some quality promoting practices, but they do not fully support quality characteristics as defined by the ISO/IEC 25010 (ISO 2010) quality standard.

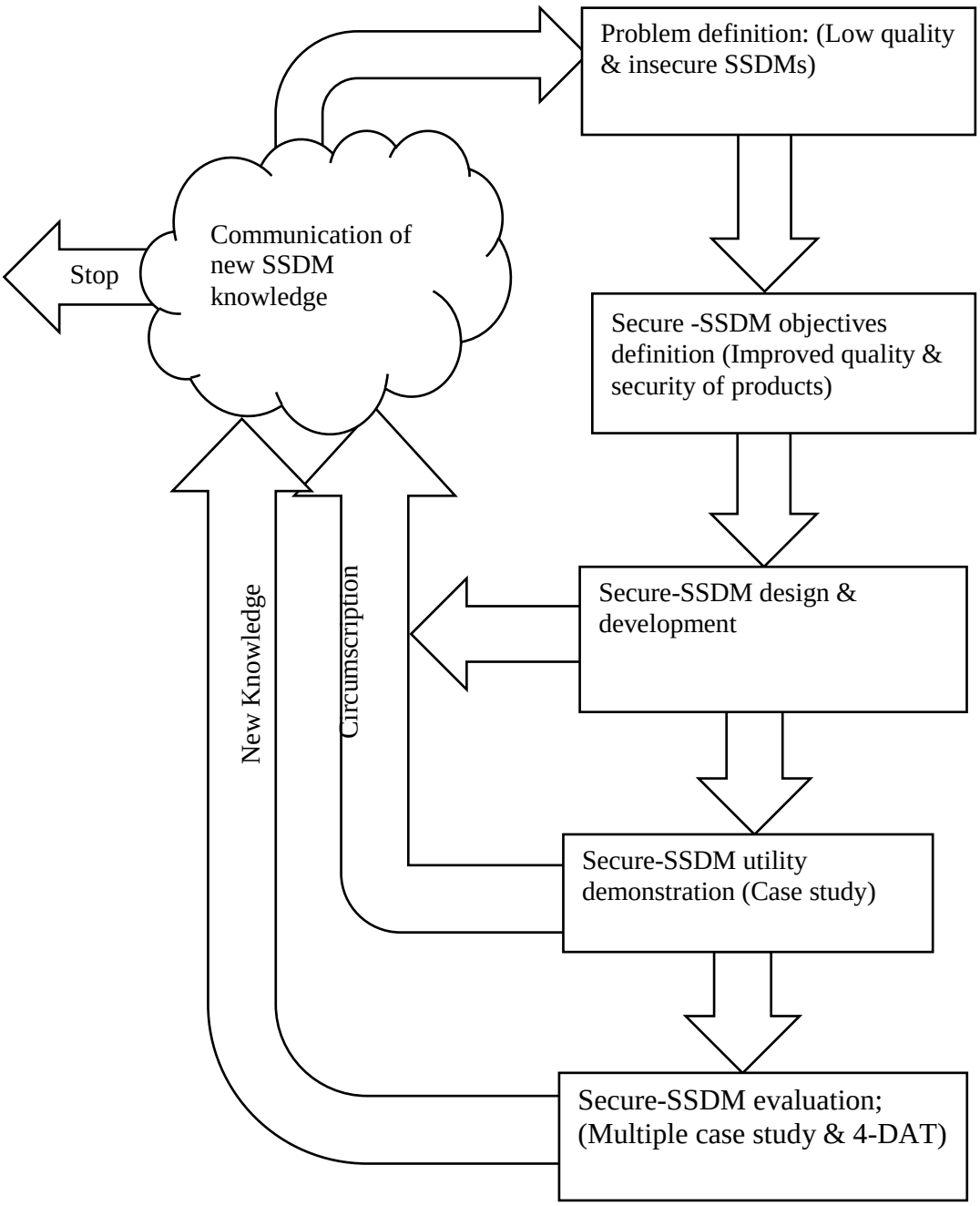


Figure 3.1: Knowledge flows in DSR (Vaishnavi, Kuechler & Petter 2017)

The mapping of the quality theory derived from the meta-synthesis against the ISO/IEC 25010 quality model revealed partial support of the quality theory for the quality characteristics defined by the standard. The derived quality theory was shown to fully support functional suitability, maintainability and satisfaction, while it partially supports usability and reliability. Performance efficiency, compatibility, security, effectiveness, portability and efficiency are not supported. At the analysis stage in Chapter 4 of this research, those quality characteristics that are supported by the SSDM framework are fully explained based on the definitions in the ISO/IEC 25010 quality standard.

3.3.2 Defining solution objectives

This research proposes a higher quality SSDM that promotes quality and security in the developed software product. The derived theory and literature review findings show that existing SSDMs have limited support for quality, and have no support for product security. The proposed SSDM should support both quality and security in the designed product. Using existing SSDMs quality practices as a baseline, the proposed Secure-SSDM builds onto these by incorporating security promoting practices derived from lightweight secure software development methods.

To encourage its uptake among solo developers, the Secure-SSDM is designed to be an agile method. This means that it is designed to be compliant with the twelve agile principles (Fowler & Highsmith 2001; Beck et al. 2001).Thus it is designed to:

i. Satisfy the customer through early product delivery.
ii. Incorporate requirements change throughout the development process.
iii. Deliver working software frequently, preferably in short cycles.
iv. Promote continuous customer involvement.
v. Motivate and empower the development team.
vi. Uphold face-to-face communication among team members.
vii. Measure project progress using working software.
viii. Uphold a sustainable development process.
ix. Focus on technical and design excellence.

x. Ensure maximum simplicity.

xi. Encourage self - organisation of teams.

xii. Allow teams to reflect on performance and adjust processes accordingly.

It should be noted that the team size for the Secure-SSDM is one, excluding the customer. This means the principles pertaining to team environments should be handled as such. The main contribution in this thesis is promoting security in the developed software products. Software security is an important quality characteristic expected of software products, especially for those that are deployed on the World Wide Web (Uikey 2015, p.28).

3.3.3 Designing and developing the proposed artefact

In the adopted DSRM, an appropriate method is used to design the artefact at the design stage. Based on the expression "Software processes are software too" (Osterweil 1997, pp.356 -357), the Secure-SSDM was designed incrementally and iteratively, characteristic of agile design. In the first iteration, quality practices drawn from existing SSDMs were used to form the primary Secure-SSDM. The primary Secure-SSDM was designed through synthesizing existing SSDMs giving the resulting methodology greater quality capabilities than the existing methods (Cruzes & Dybå 2011, p.443). Peffers *et al.* (2008) used a similar approach in designing the DSRM used in this thesis. In their case, method practices were extracted from existing DSR methods to form the core method practices, thus giving the methodology a firm grounding (Peffers et al. 2008, p.52).

The first iteration in designing the Secure-SSDM, was dedicated to formulating the primary framework. This primary framework was initially presented at a postgraduate seminar, and feedback from the participants was used to refine the methodology. Further, to ensure rigour in the method design cycle, the process of building the framework, together with the resulting framework were presented at an international peer-reviewed conference. This conference publication is detailed in Moyo and Mnkandla (2019).

In the second iteration of the design cycle, lightweight security practices were derived from secure software development processes. The latter provided the best alternative source as the literature review had conclusively shown that existing SSDMs do not have security promoting practices. An algorithm adapted for the purpose was used to integrate the security practices to the primary SSDM.

3.3.4 Demonstrating the utility of the artefact

The Secure-SSDM is designed to build quality and security in the developed software. Demonstrating its utility entails using the methodology to design and develop quality and secure software products. A multiple case study was used for the purpose. The first case study was carried out in an academic institution. Thirty-nine undergraduate students pursuing a computer science degree participated in the study. To undertake the evaluative case-study, the researcher first sought for and was granted ethical clearance by the participating students' institution. In addition to the participants' institution clearance, an overall ethical clearance for the study was sought for and granted by UNISA. The ethical clearance and the clearance letter from the university gate keeper are attached in Appendix A.

After receiving both clearances, an invitation to participate was extended to all Computer Science second - year students. These were students enrolled at the National University of Science and Technology (NUST), Zimbabwe for the 2018-2019 academic year. Thirty-nine students opted to participate. Participants were asked to apply the Secure-SSDM in developing individual sized software projects to address industry needs. Mini projects were undertaken in the areas of Education, Business, Health, Environment and Government. This lot of students was found suitable for this purpose due to the fact that the researcher had access to them. Further, the students had undertaken courses necessary for software development. The detailed description of the case study is given in Section 6.3.

At the onset of the case study, the roles, tasks and deliverables from the methodology were explained to the participants. The expectations from the study were not explained so as to minimise bias (Pohl & Hof 2015). While the use of the methodology by student participants provided a means for formative evaluation, it further provided a means for eliciting method requirements from a developer perspective. After the students had used the method, they were asked to comment on the usability and appropriateness of the method for building quality and secure software systems. Class discussions were conducted to obtain feedback from the participants. Documentation to support the designed software products was reviewed to establish methodology execution by the participants. The comments obtained from the students after applying the methodology were used to generate knowledge in the circumscription cycle.

The second case study involved industry developers applying the methodology to develop web-based software applications of their choice. Development was however not restricted to

web applications only. The web-based applications were chosen to demonstrate the attainment of quality characteristics expected of the software products built using the SSDM. Web-based applications are expected to be secure, simple to use, consistent, understandable, reusable, maintainable and complete (Sfetsos et al. 2016¸ pp.1-2; Uikey 2015, p.28). The web-based applications were found to be ideal to cover all the expected product quality characteristics developed using the Secure-SSDM. The demonstration section of Chapter 6 gives a detailed description of an example case system designed to demonstrate the utility of the Secure-SSDM.

3.3.5 Evaluation

Evaluation checks how well the designed artefact addresses the initial artefact objectives. It is also a process of checking the usability, usefulness and efficiency of an artefact (Venable et al. 2016, p.77). Evaluation is an important aspect of DSR, and rigorous methods should be applied to evaluate the designed artefact (Hevner et al. 2004, p. 13; Venable et al. 2016, p.77). To ensure rigour in the evaluation process, two forms of evaluation were applied in this thesis. A theoretical evaluation was performed to check the compliance of the Secure-SSDM with the requirements of agile methods as defined in the agile manifesto. The 4-DAT framework (Qumer & Henderson-Sellers 2006; Qumer & Henderson-Sellers 2008) was found ideal for the theoretical evaluation purpose. Other researchers (González-Sanabria, Morente-Molinera & Castro-Romero 2017; Leppa 2013; Ghani et al. 2014) have also used the framework for the same purpose, proving its utility for the purpose.

The four dimensions used to evaluate methodologies in this framework are method scope, method agility, agile values characterisation, and software process characterisation. Method scope considers the project and team sizes, development and coding styles, technology and physical environments, and business and abstraction culture of the artefact. Method agility evaluates the method practices and stages against the agile characteristics of flexibility, speed, leanness, learning and responsiveness (Qumer & Henderson-Sellers 2006, p.504). Each of the method practices and method stages is assessed for exhibiting these characteristics. A practice is assigned a score out of five, depending on the presence or absence of these. The highest score is five (5), if all are present and the lowest is zero (0), if none of these exist. Agile values characterisation identifies those practices in the agile method that support agile values. The authors define six values necessary for the purpose which are: individuals and interactions over processes; working software over comprehensive documentation; customer collaboration

over contract negotiation; responding to change over following a plan; keeping the process agile; and keeping the process cost effective (p.505). The evaluation process sought to identify practices within the proposed methodology supporting these values. Lastly, method characterisation identifies processes within the method that support: software process life cycle coverage; project management support; software configuration management; and process management support (p.506).

The main aim of this theoretical evaluation was to determine the agility of the methodology using an established model. Since this thesis proposes a lightweight agile methodology for use by solo developers, it was necessary to evaluate this characteristic of the Secure-SSDM. The theoretical evaluation process is detailed in Section 6.7.1.

A multiple case study was also used to demonstrate the utility of the Secure-SSDM, as well as to evaluate the usability, effectiveness and completeness of the designed methodology in its intended environment. Oates (2006, p. 116), recommends that software engineering artefacts be evaluated in a real- world environment. Case studies are appropriate for empirical evaluation when the boundary between the artefact under study and its context are unclear (p.142). This is true for software development methodologies whose success is influenced by the people and the environment in which they are used (Runeson & Höst 2009, p. 137). In DSR evaluation is a continuous process whose output feeds back to the design process (Hevner et al. 2004). The application of the methodology in developing software products by both student and industry developers served to demonstrate and evaluate the usability of methodology by the target community.

Various measures were put in place to address threats to validity associated with case studies. Threats to validity in case studies can be internal, external, and construct or can be threats due to reliability (Baca & Carlsson 2011, p 152 - 153). Internal validity pertains to the unexpected influence by another factor on the factor under investigation in causal relationships (Runeson & Höst 2009, p.154). In this research this would mean an outside factor induces quality and security in the software products, besides the practices embedded in the Secure-SSDM. External validity pertains to the extent to which the results of the case study can be used in similar cases. Construct validity refers to the extent to which metrics of measure evaluate the aspects being considered in the case. Lastly, reliability refers to the repeatability of the study to give similar results.

To deal with the threat to reliability, a case study protocol was designed to guide the study and ensure data collection transparency (Yin 2015, p.198). In addressing external validity, multiple case studies were conducted, the student case study and the industry cases. To address the construct validity threat, for the academic case study, data was collected through focus group discussion and document review of strategic models in the SSDM cycle. These catered for data and method triangulation. An interview guide was also developed for use with the industry participants. A theoretical framework describing causal relationships between quality promoting practices and quality characteristics was used to deal with the internal validity threat.

The development of web – based applications was considered a good representation of all the other forms of software applications as it enabled the researcher to assess the capability of the methodology to facilitate the development of a software product with all the targeted quality characteristics. Web-based applications are expected to be secure, simple to use, consistent, understandable, reusable, maintainable and complete (Sfetsos et al. 2016, pp.1-2; Uikey 2015, p.28). The case study projects therefore test the capability of the methodology to develop software with the characteristics set in the suggestion step. The study protocol given in the following sub-section 3.3.5.1 explains how the case study was conducted.

Case Study Protocol (Plan)

To ensure a high quality case study, Runeson and Höst (2009 pp.138 - 140) suggest the formulation of a plan with the following content: Objective; The case; Theory; Research questions; Methods and Selection strategy. Yin (2015 p.199) refers to this plan as a case study protocol. The objective spells out the reason for undertaking the case study. The theory defines the context of the case study, and the researcher has adopted the quality theory derived from this study for the purpose. The research questions help to shape the objective set for the case study. The data collection methods define how data is collected from the case while the selection strategy identifies points and sources of data collection.

Objective — The objective of this case study was to establish the perceptions of independent (solo) developers on the usefulness of the quality practices embedded in the Secure-SSDM in building quality and secure software.

The case —A multiple case study was conducted with solo developers. The first set of developers was made up of thirty-nine student participants in a university setting. The second

set of participants consisted of three industry developers working independently in individual sized projects. The independent developers were experts in solo software development, each with an average of four years developing software independently.

Theory— Based on the SSDM framework derived from the literature, and the security practices drawn from secure software development methods, the Secure-SSDM has quality practices that support the quality characteristics in the developed software product. Table 3.2 shows the quality and security practices built into the methodology and expected quality characteristics in the resulting product. The quality practices shown in the table promote the delivery of software products with quality characteristics defined in the corresponding column. As an example, test driven development, refactoring, unit testing and use of a version control system, promotes maintainability of the software product. According to the ISO/IEC 25010 quality model, a maintainable product is one that is modular, reusable, analysable, modifiable and testable.

Table 3.2: Quality practices, associated product quality characteristics and sub-characteristics

Quality Practices	**Anticipated Impact on Product Quality Characteristics**
Use of Development standards Use of Small user stories & tasks Automated code review Code refactoring Design of testable code	Product simplicity
Adoption of developmental standards Product validation	Consistency
Use of simple metaphors Test driven development Unit testing Use of a dummy partner/ self-criticism	Understandability Code quality
Simple module design Test driven development Refactoring Unit testing Task automation Version control system	Maintainability
Use of a work break down structure Creation of product backlog Use of a product checklist	Product completeness
Security awareness training Use of use misuse case diagrams Adoption of security standards Security test design Security testing	Security

Research questions—research questions help to deliver the defined objectives. To that effect the following questions guided the case study:

- *What are the perceptions of solo developers regarding the use of practices and stages of the Secure-SSDM in building quality and secure software?*
- *What are the perceptions of student and industry developers regarding the integration of security practices into the Secure-SSDM?*
- *What improvements can solo developers suggest in each of the Secure-SSDM methodology stages?*

Data collection methods— In both the academic and industry case studies, more than one method was used for data collection to obtain the perception of the developers on the utility of the Secure-SSDM. To ensure reliability of the data collected from the student case study, focus group discussions together with document analysis were used to triangulate the data collection process (Yin 2015, pp. 197 - 198). At the end of the case study, a focus group discussion with student participants was conducted to obtain their views on the utility of the methodology in building quality information systems. The focus group was designed to fit within the two-hour period allocated to the class sessions of the students.

In this case study the focus group discussion was deemed the most appropriate, compared to interviewing the participants individually, due to the large number of students, and the short duration of the semester. The focus group discussion also provided for checks and balances on the views pertaining to the utility of the methodology from this set of participants (Runeson et al. 2012). In a number of situations, the students helped clarify and correct each other's perceptions on some practices. Since the researcher was in charge of the class, it was also easy to direct the group towards the most important aspects of the discussions, without interfering with the outcome of the discussion. A teaching assistant helped with the data collection from the discussions. The assistant provided some form of researcher triangulation. The focus group guide used with participants is given in Appendix B, and the data collection template for the focus group is given in Appendix D.

The student participants were also asked to submit documentation accompanying their projects. This is a normal practice for all mini projects carried out in this academic

environment. The documents submitted were analysed for intermediate models expected with each system component. They were used to confirm the students' comments on the utility of methodology. On the methodology section of project documents, students normally comment or give reasons why they use a certain methodology in developing software. This section was used to gather the perceptions of the student on the Secure-SSDM.

In the industry case study, face to face semi-structured interviews were used with two of the participants. For the third participant, video-conferencing was used as the participant had changed cities at the time of the interview. Member checking and feed-back were used to confirm the data collection and interpretation of the participants' opinions. Cross-case analysis was used to analyse the data collected from the two case studies. This entailed first analysing the data from the two case studies separately, after which the data from the two was analysed through checking of similarities and differences.

Participants selection strategy

In the academic case study, student developers were selected intentionally (Runeson & Höst 2009, p. 140). The selected class had been taught requisite courses for software development. Among the courses that the selected class of students had covered were: Systems Analysis and Design; Object-oriented Software Concepts and Development; Software Design Methodologies; Internet and Web Design; and Societal Computing. These five courses of the second year of these participants, are highlighted in this thesis, due to their relevancy for this case study. The detail of what is covered in each of the courses is discussed in Section 6.3.

For the industry case study, participant A was recruited by the researcher from their previous interaction in solo development projects. The participant was a university employee, whom the researcher had previously worked with in developing software for clients. During that period, the participant had done several individual projects on a consultancy basis. While a full-time developer at the university, during their free time, they worked on their independent projects. This participant was selected for their expertise in software development, and in particular on solo projects. Participant A was asked to refer other solo developers to the project. Two other participants were identified through this snowballing process, bringing the total to three industry developers. The full credentials of the developers who participated in the industry case study is discussed in Section 6.4.

3.3.6 Communicating the results of DSR research

Communication adds knowledge to the SSDM and SSD environments. In this thesis two academic seminars, one peer reviewed conference and one peer-reviewed journal were used as channels to communicate the design process and the evaluation results of the Secure-SSDM. Of the two post-graduate academic seminars, the first seminar was used to communicate the proposal to develop the artefact, and the second seminar was used to communicate the primary Secure-SSDM framework. Participants in both seminars gave feedback that helped to shape the artefact. An international peer review conference was used to communicate both the quality theory and the preliminary SSDM framework derived from the existing SSDMs. These are discussed in Moyo and Mnkandla (2019). The design process and the resulting final version of the Secure-SSDM was published in an international journal (Moyo & Mnkandla 2020).

Figure 3.3 summarises activities of the DSRM as carried out in this research. Chapter 1 was dedicated to defining and scoping the research problem. Chapter 2 reviewed the SSDM literature helping to refine the research problem, and initiated the generation of the quality theory, which is concretized in Chapter 4. Chapter 4 details the Secure-SSDM quality theory and objectives, setting measures for evaluating the methodology. Chapter 5 discusses the design and implementation of the proposed artefact using appropriate techniques and tools. Chapter 6 demonstrates the utility of the Secure-SSDM in designing and implementing high-quality and secure software products.

3.4 Conclusion

Subsections 3.1 to 3.3 have outlined the work done, how it was done and when it was done. This outline is summarised in Figure 3.2. The paradigm that guided this research is DSR. This was deemed applicable as it allowed the researcher to design the Secure-SSDM iteratively, with each iteration improving on the quality of the methodology. The interpretivist philosophy was also deemed appropriate as a complementary paradigm to enable the demonstration of the methodology in a live environment. Software development methodologies are better studied in their context as their success is influenced by humans and the environment in which they are used. The designed methodology was evaluated theoretically using the 4-DAT framework and empirically using case studies, both in academia and industry. Document sampling was the main data collection method at the problem definition stage. Semi-structured interviews, document sampling and informal observations were also carried out on the student participants

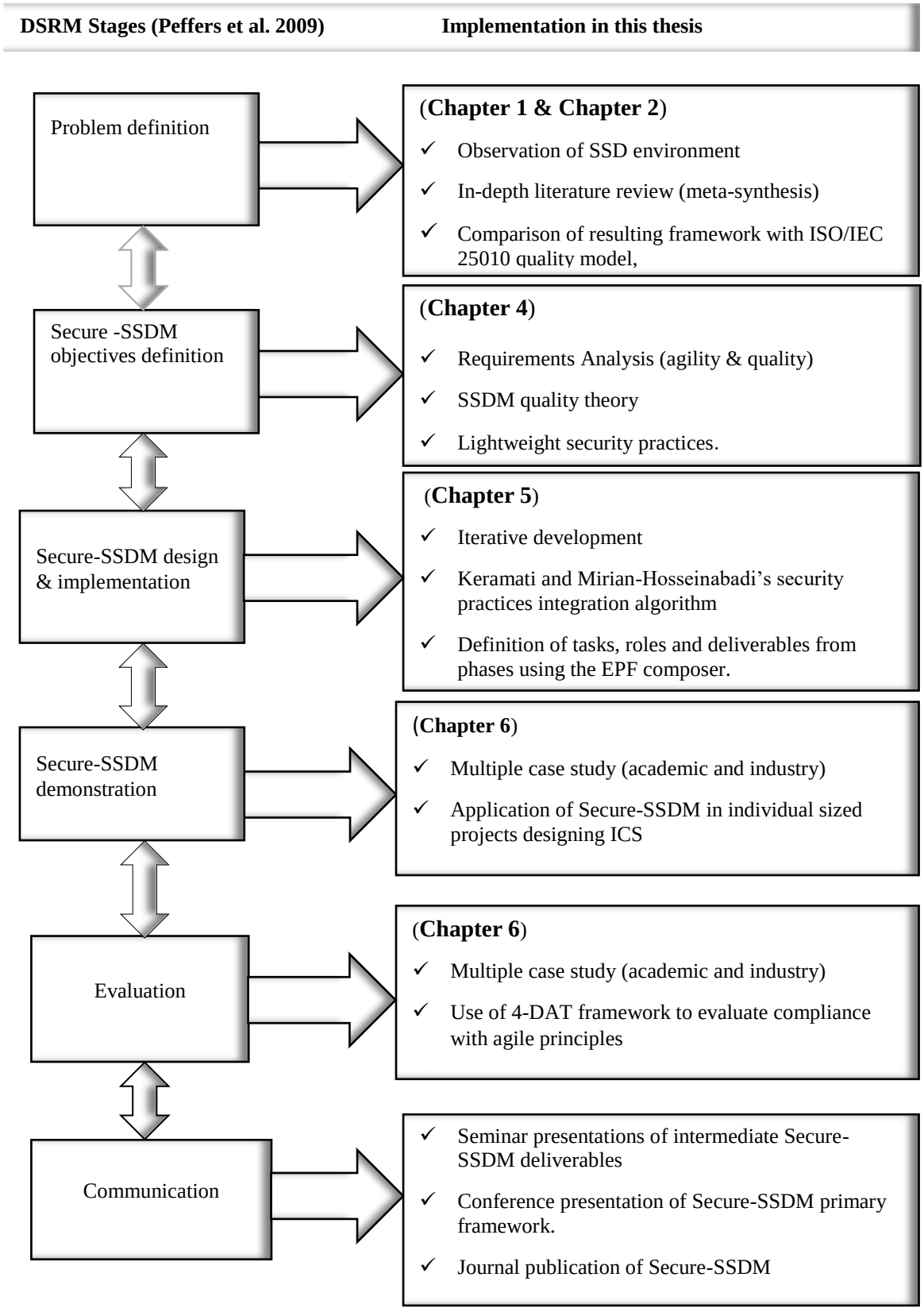


Figure 3.2: Using DSRM to design the Secure-SSDM (Adapted from Peffers et al. 2009)

using the method. Data collected from the interviews and system documents was analysed qualitatively.

3.5 Chapter Summary

This chapter outlined the research paradigm on which this research is premised. DSR is most favourable for artefact design. For data collection, an interview guide was presented and data analysis methods discussed. The next chapter gives and in-depth analysis of the Secure-SSDM activities, tasks and roles. The chapter carries out an in-depth analysis of the existing SSDMs with the aim of defining objectives for the Secure-SSDM.

CHAPTER 4 SECURE-SSDM REQUIREMENTS ANALYSIS

4.1 Introduction

In Chapter 3 a research methodology was developed to guide this study. Two philosophical paradigms were adopted, the design science research (DSR) and the Interpretivist paradigms. The latter enabled the derivation of perceptions of the developers regarding the desired utility of the software development methodology (SDM) under design. These perceptions were collected from solo developers who participated in two different case studies designed to evaluate the utility of the Secure-SSDM in a live environment. DSR was deemed appropriate for the design of a satisficing artefact architecture to improve the quality and security of software products of an identified community (Peffers *et al.* 2008). The target community in this case, is the solo developers. From a Software Engineering perspective, artefacts can be in the form of algorithms, methods or techniques (Wieringa & Daneva 2015), among others. The term method here is used interchangeably with methodology.

The research methodology designed in Chapter 3 provided a scientific grounding (Mnkandla 2016, p.33) for this research, at the same time promoting research rigour in both the design and evaluation of the resulting artefact (Hevner *et al.* 2004, p.83). Following closely the research methodology steps presented in Section 3.3, this chapter presents an in-depth analysis of the SSDMs identified from the literature, in view of creating requirements for the Secure-SSDM. The chapter revisits the SSDMs identified in Chapter 2, and the derived framework to analyse its suitability for building high-quality software.

Requirements discussed in this chapter are drawn from the literature and from solo developers who participated during the formative evaluation of the methodology. The in-depth review of the literature constitutes the rigour cycle of the DSR process (Hevner *et al.* 2004) meant to position the Secure-SSDM within the present literature (Barafort et al. 2018, p.28). In DSR, both the design process and the artefact under design evolve during the design and evaluation processes (Hevner *et al.* 2004, p.78). The two case studies and the continued review of the literature during these processes contribute to the evolution of both the SSDM knowledge base and the Secure-SSDM.

The following sections detail the methodology requirements analysis process. Section 4.2 outlines requirements in general and their importance in artefact design. Section 4.3 analyses the identified SSDMs, discussing their quality practices, and how these promote quality in the

developed product. Section 4.4 details the security promoting practices identified from the literature. Section 4.5 outlines the Secure-SSDM requirements. Section 4.6 summarises the requirements analysis, giving a summary of the objectives expected of the Secure-SSDM, and characteristics expected thereof.

4.2 Requirements

Requirements are characteristics or capabilities to be portrayed by an object under development (Garg 2017, p. 64). In DSR, requirements provide a means for evaluating the usefulness, efficiency and quality of the artefact under design (Hevner *et al.* 2004, p.85). In the context of the DSR methodology adopted in this research, requirements constitute solution (or artefact) objectives. The aim of this research was to develop a lightweight, high-quality, and secure software development methodology for use by solo (freelance) developers. The resultant Secure-SSDM from this research should therefore exhibit characteristics in line with this aim. This chapter elaborates these characteristics (system objectives) enabling the formulation of a befitting design in Chapter 5. The methodology characteristics were drawn from a systematic literature review of the lightweight SSDMs and the review of lightweight secure software development methods. Identified methodologies and practices were reviewed individually, focusing on each item's promotion of quality in the developed software product. These existing methodologies' objectives helped to establish objectives for the methodology under design.

4.3 Analysis of the Existing Lightweight SSDMs

Existing lightweight SSDMs form the knowledge base from which this study draws methodology practices. They also helped to derive means for designing the proposed SSDM (Hevner et al. 2004, p.80). Existing SSDMs therefore provided foundations on which the proposed methodology is built. It is important therefore that this research consistently searches this knowledge base for the purposes of identifying any updates or new tools and materials released into this valuable source so that both the design process and artefact under development are kept current. A revised search of the literature databases publishing software engineering research conducted after the first search identified one more publication (León-sigg *et al.* 2018) to bring the total complement to seven. The following is a comprehensive list of solo software development methodologies identified by this research:

i. Freelance as a Team (Faat) (Bernabé, Navia & García-Peñalvo 2015),

ii. Personal Extreme Programming (PXP1) (Agarwal & Umphress 2008),

iii. Personal Extreme Programming (PXP2) (Dzhurov, Krasteva & Ilieva 2009),

iv. Go – Scrum (Ramingwong, Ramingwong & Kusalaporn 2017) ,

v. Scrum Solo (Pagotto *et al.* 2016),

vi. DeSoftIn (González-Sanabria, Morente-Molinera & Castro-Romero 2017) and

vii. MIDS Adaptation (León-sigg *et al.* 2018).

The publication dates of the SSDMs indicate that research in lightweight solo software development is still ongoing. The last five years have seen the publication of the majority (five out of seven) of the articles found. The research output in the areas is however still low. Due to limited research in SSDMs, the literature search was extended to include studies that discuss solo software development, even though the publications were not formulated as software development methods. These include articles by these authors: Dent 2008; Hollar 2006; Raymund et al. 2005; and Wesslén 2000. This was done to enrich the new SSDM with quality practices from the existing peer reviewed literature (Nwasra et al. 2016, p.70). In the following subsections the analysis of the identified publications is presented in the order of their listing.

4.3.1 Freelance as a Team (Faat)

Faat is an agile methodology introduced by Bernabé, Navia and García-Peñalvo (2015). The methodology integrates agile practices ideal for individual development. Drawn from eXtreme Programming (XP) (Beck 2000) and Scrum (Schwaber 1997), the practices in Faat are divided into strategic, workflow and auxiliary practices.

Strategic practices equip the developer with skills to make the best option when faced with several options during the development cycle. These can be summed up as simplicity, embrace change and make decisions (Bernabé, Navia & García-Peñalvo 2015, pp.687-688). Simplicity according to these authors means that the developer should always choose the simplest option when faced with a decision. This minimises exerting effort on activities which might require later changes as developers respond to user preferences. At the same time while accepting

change, the developer is encouraged to make necessary decisions when decision making calls, so as to avoid doing work outside the project.

Workflow practices describe developmental activities with their associated deliverables. They constitute: User stories; Estimation; Planning; Development; Review and Iteration close (Bernabé, Navia & García-Peñalvo 2015). The initial practice is dedicated to the creation of user stories**.** User stories describe expectations of users from the system (Bernabé, Navia & García-Peñalvo 2015), and are a popular feature of agile methods. They are a simpler way of defining functional requirements. Each identified user is expected to specify their expected functionality from the software product, together with the value obtained from that functionality. Using the INVEST acronym, user stories are formulated to be independent of each other, to be negotiable, to be of value to the user, to be estimable and small enough to be tested independently (Bernabé, Navia & García-Peñalvo 2015; Lucassen *et al.* 2016). The INVEST acronym enables effective communication of software requirements between the developer and the users (Wake 2003). A fully formulated user story should have a unique identity, title and description, associated acceptance criteria, priority and should belong to an appropriate class (Bernabé, Navia & García-Peñalvo 2015, p.688) .

Most authors (Agarwal & Umphress 2008; Bernabé, Navia & García-Peñalvo 2015; Ramingwong, Ramingwong & Kusalaporn 2017) from the reviewed literature concur on the effectiveness of user stories as a requirements elicitation technique. User stories are an ideal practice to incorporate into the methodology under development. The INVEST acronym provides a means of building portability into the designed software.

Continuing with the analysis of Faat, once the user stories are identified, **t**heir estimated duration, together with the required resources, are projected. For a lone developer, estimation is recommended to be done in hours. Projecting completion time in hours ensures that the tasks are kept small enough to enable exact estimation. Estimation is expected to be a continuous process which improves with time as the developer compares the exact time of completing tasks with the projected, and adjusts future tasks accordingly.

Using information obtained so far, the next practice creates a prioritised list of user stories indicating the time in hours, value to be obtained from each story, and condition (s) for acceptance of the user stories. This prioritised list is known as a product backlog. Tasks that deliver defined value from each user story (or a collection of stories) are then organised into a sprint. A sprint duration of thirty-two to thirty-five hours (at most two weeks) is

recommended to deliver value to the user. Functionality to be delivered by the sprint determines the objective of that sprint.

A developer using Faat adopts a version control system to manage their source code during the development phase. A version control system helps to keep track of changes in code, enabling the developer to fall back to the last stable code in the event that a change results in unstable code. In addition, test driven development is the recommended approach, together with refactoring for large systems. Once a user story passes the test, the developer is advised to compare the estimated time against the planned, and use that to revise estimates for remaining user stories. This is a practice also recommended by other authors (Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009). The review follows development. Here the developer looks back at the work done in the sprint and evaluates this against the planned. Automated tools such as integrated development environments (IDEs) can be used to check for code quality and corrections done accordingly (Bernabé, Navia & García-Peñalvo 2015, p.693).

Iteration close marks the end of the sprint. A working module is integrated to the system under development using the version control system, and set ready for installation at the customer's site. If it is the last sprint, then this marks the end of the project.

4.3.2 Personal Extreme Programming (PXP1)

PXP1 is a scaled down version of eXtreme Programming (XP) that has been hybridized with the Personal Software Process (PSP). The literature search conducted in this research found two publications by different authors with the same name. This study uses numbers to distinguish the two methods. PXP1 discussed in this session is the publication by Agarwal and Umphress (2008), while PXP2 discussed in Section 4.3.3 is the publication by Dzhurov, Krasteva and Ilieva (2009). PXP1 is an incremental, iterative process which incorporates quality practices from the two methodologies that it is based on. It exhibits most of the quality practices of XP. These are: the use of metaphors and user stories; use of small system releases; simple designs; test driven development; refactoring; continuous integration and the adherence to appropriate coding standards (Agarwal & Umphress 2008, p. 85). Its stages of Planning, Development and Post-mortem ensure simplicity of the process. Activities and associated quality practices executed in these three stages are summarised below;

Planning – The developer begins the project by establishing system requirements. This is done through the creation of a system metaphor easy enough to be understood by the developer. At the same time user stories are acquired and written on small cards. Each card carries the story described in simple language, its associated priority and cost. Some authors (Lucia & Qusef 2010; Sillitti & Succi 2006), recommend the use of the customer's language for simplicity. Prioritisation of user stories is a key feature in agility (Heck & Zaidman 2018). It ensures that the most important functionality to the user is delivered first. User stories are then broken into features which are organised into feature sets, after which a design is created for each feature set. Planning culminates in an iteration schedule indicating how the feature sets will be implemented.

Development – During development, the developer works on the feature set, starting with high priority features. From features, tasks are created which are then sorted according to priority. The developer picks tasks from the priority list, formulates unit tests for each task and writes the code for the task under development. This is a concept of test-driven development. The concept is a well-supported practice of software quality (Crispin 2006; Abrantes & Travassos 2011; Fitzgerald & Stol 2017; Rafique et al. 2013; Sfetsos & Stamelos 2010) . Test-driven development and unit testing enhance code quality, while refactoring promotes system extensibility and maintainability (Bernabé, Navia & García-Peñalvo 2015, p.690; Dzhurov, Krasteva & Ilieva 2009, p.258).

To further improve code quality, the developer performs code walk-throughs for each task, before unit testing. Unit testing is followed by successive acceptance and integration testing, after which the successfully implemented task is integrated into production code. Where the developer is also the customer, Agarwal and Umphress (2008) recommend that the developer carries out dialogue with himself during acceptance testing. This is true for systems that are developed for personal use or for general purpose. The practice of self-dialogue is similar to that of the use of a dummy companion to review code (Bernabé, Navia & García-Peñalvo 2015). It mimics the practice of peer-review which is missing from a solo development environment. Throughout the development process, versions of code are maintained to enable a smooth fall-back to the last stable state of the system. The developer maintains development, refactor and production code baselines to enable high-quality code (Agarwal & Umphress 2008).

Post-Mortem- During this stage the main aim is to perform acceptance testing of the system as a whole. The tested code is integrated into the production baseline code. Two brief stages complete PXP1; the Entry stage where the developer adopts appropriate coding standards for the system under development and the Exit stage, where the output is a fully tested system integrated into the production code baseline.

4.3.3 Personal Extreme Programming (PXP2)

Like the PXP1 discussed in Section 4.3.2, PXP2 is a hybrid of XP and PSP. The aim in PXP2 is to improve the product quality of autonomous developers at the same time improving their development performance in the software market (Dzhurov, Krasteva & Ilieva 2009). In hybridising PSP with XP, the authors' intention was to reduce documentation associated with PSP and thus produce a methodology that could readily be adopted by solo developers. Designed to be iterative, PXP2 facilitates response to changes throughout the software development process. At the core of PXP2 is automation of recurring processes to improve developer productivity.

A PXP2 project begins with the stage Requirements. Presented as an optional phase in the methodology, the developer establishes both forms of the system requirements, functional and non-functional requirements. The assumption made is that requirements are static, and that in the case of any changes, these should be reflected in the requirements list, and planning revisited (Dzhurov, Krasteva & Ilieva 2009).

Once requirements have been established, Planning is carried out for the whole project based on the requirements established in the previous stage. The developer starts the planning phase by adopting a development language and a platform appropriate for the product under development. From the requirements list, the developer then derives tasks to be undertaken. Identified tasks are categorised, at the same time providing time and cost estimates for these, based on previous estimates of similar task categories (Dzhurov, Krasteva & Ilieva 2009). Tasks are recommended to be small enough to facilitate accurate estimates.

Scheduled to last for one to three weeks, the stage Iteration Initialisation follows planning. An iteration is designed to deliver a version of the product developed from tasks selected for the iteration. Bernabé, Navia and García-Peñalvo (2015) recommend a similar period for iteration duration of not more than two weeks. These authors (Bernabé, Navia & García-Peñalvo 2015; Dzhurov, Krasteva & Ilieva 2009; González-Sanabria, Morente-Molinera & Castro-Romero

2017) from the reviewed methodologies concur in keeping the iteration period short enough to keep the developer focused.

During the subsequent phase of Design, the developer creates a design appropriate for the requirements under development. Simple designs are recommended to avoid the developer working on tasks that do not add value to the current tasks under development. To ensure simplicity in design the developer may make use of tools familiar to them.

Implementation translates the design into a deliverable. For each task, unit tests are carried out on developed code, identified defects removed before integration and acceptance testing. The authors recommend the use of automated development tools to perform quality tests such as code coverage of unit tests.

At the System Testing phase, developers test the whole system for adequacy in meeting user requirements. This is a key feature in software quality. Any defects identified are fixed and recorded. The defects record serves as reference for future projects, and gives hints on sources of defects. The last stage, Retrospective, serves as a point of knowledge management in the development process. The developer is advised to collect data associated with the process for future use. Data collected at this stage enable more accurate estimates in coming cycles or projects.

This study noted a lot of similarities in the two PXPs. This is not surprising as the two methods, though designed by different authors, both draw their core practices from XP and PSP. It is also interesting to note the emphasis of these methods on keeping the development iterations short as a means of encouraging productivity. Besides encouraging productivity from the developer, short iterations enable development process visibility, subsequently encouraging product acceptability. In the following section, a slightly different methodology, Go-Scrum, is detailed. Go-Scrum differs from these in that it is based on Scrum practices (Schwaber 1997).

4.3.4 Government -Scrum (Go – Scrum)

Go-Scrum, also known as Solo-Scrum, is a scaled down version of Scrum, comprising of those practices that are executable by a single developer (Ramingwong, Ramingwong & Kusalaporn 2017). Go-Scrum is designed for use in bureaucratic organisations such as government departments. Quality practices in Go-Scrum include: the use of a kick-off meeting at project onset; the use of story cards to capture user requirements; creation of a product backlog in

collaboration with the user; and the use of a work break down structure to capture product components; just to mention a few. The stages in Go-Scrum are overviewed in the following paragraphs:

Management Buy -in – This stage is dedicated to educating the users on the development process. In this methodology educating users on the development process is viewed as a means of encouraging their participation on the development process. It is unique for this methodology, perhaps meant to address the bureaucracy associated with large organisations. Apart from educating them on the development process, users are informed of the product components and deliverables associated with the development process. This provides check points for both the developers and project stakeholders.

Kick-off Meeting and Story Discovery – Stakeholders of the software under development meet to discuss requirements of the system. Meetings arranged early in the development cycle help to shape project progress (Heck & Zaidman 2018). In the meeting, each stakeholder submits their requirements in the form of user stories captured on small story cards. The success of the kick-off meeting and the associated requirements discovery, is heavily dependent on the ability of the developer to encourage participation among all stakeholders so as not to miss any requirements.

Project Planning – Based on user requirements collected in the previous stage, the developer creates a prioritised product backlog with the help of the user. The product backlog is a key artefact in Scrum. A backlog from the view of Scrum is a product functionality, defect, bug or any aspect of the software that is outstanding (Schwaber 1997, p. 15). To some extent a product backlog shows work still to be done in the project. All reviewed methodologies emphasise the creation of a product backlog at the onset of development, although this may change during the course of the development and have different terms in each methodology. In PXP2 (Dzhurov, Krasteva & Ilieva 2009, p.254) this is called a requirements document, while this is termed a feature set in PXP1 (Agarwal & Umphress 2008, p.83). This indicates the significance of this practice in developing quality products.

Release and Sprint Planning – A release results in the installation of a viable component at the customer's site or developer's machine. A prioritised sprint backlog is created from the product backlog. A number of authors (Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; Ramingwong, Ramingwong & Kusalaporn 2017) concur on the importance of backlog prioritisation or on the prioritisation of user requirements (Agarwal & Umphress 2008;

González-Sanabria, Morente-Molinera & Castro-Romero 2017). Go-Scrum recommends the use of function points as an estimation technique to gauge effort required in any sprint. In function point estimates the developer considers such aspects as the input, output, processing, and sizes of files associated with the required component under estimation. This information can be derived from a quick sketch of the relationship of the component under consideration with the rest of the software components.

Sprint – A simple design is created for tasks in the sprint to get the sprint rolling. For the tasks under development, burn down charts are used to show task progress. These indicate work performed, work in progress and work to be performed for a task. This helps the developer not to miss any task functionality at the same time visualizing development progress. To check progress with users, the developer holds at least two meetings per sprint, in place of daily meetings as per the Scrum methodology. This serves to keep users interested, particularly in a bureaucratic environment. Each sprint culminates in a sprint review that captures data on sprint progress. A sprint delivers functionality that is tested for acceptance by users. The sprint review also serves to confirm requirements to be delivered in the next sprint before embarking on the sprint.

It is clear that Go-Scrum borrows all of its practices from Scrum. Like Scrum it is developed to be flexible, constantly adhering to changes in the environment, with its success premised user involvement. If properly followed the methodology improves the quality of software products. A similar methodology to Go-Scrum is Scrum solo. The latter is detailed in Sub-section 4.3.5.

4.3.5 Scrum solo

Scrum solo (Pagotto *et al.* 2016) is a hybrid of Scrum and PSP. It is an iterative process that delivers the software product in increments. The methodology shares a number of characteristics with Solo scrum and the following paragraphs gives an overview of the phases of Scrum solo.

Requirements – At project onset, the developer collects system requirements from the customer. Requirements define the scope of the software product. From the requirements a product backlog is generated with the customer's assistance. The product backlog should indicate a list of features to be implemented, together with their dates of entry into the backlog (Pagotto *et al.* 2016). To fully understand the requirements, it is recommended that the

developer creates a prototype that can be used to verify the requirements. The prototype should capture all product backlog items, with each item represented in its screen in the prototype. Prototypes are an acceptable traditional way of understanding user requirements, particularly for complex products. In this methodology they serve as a requirements elicitation tool.

Sprint – The sprint selects priority tasks from the product backlog that are used to create a deliverable for the current sprint. The sprint backlog stores information similar to the product backlog, only that these items in the sprint backlog are those that contribute to the functionality to be delivered in the current sprint. Artefacts for the current sprint can be represented using appropriate unified modelling language (UML) diagrams. These include diagrams such as: the use case diagrams, that capture functionality to be delivered in the current sprint; sequence diagrams, to capture the flow of events in delivering the functionality; as well as class diagrams to capture the relationships among components modules designed to deliver the functionality. For data-based applications the methodology recommends the use of entity relationship diagrams, to capture and model the relationship among objects about which data is stored. The developer should use the right diagrams to indicate the type of detail in the sprint. A project repository should be created to store these diagrams. Further, sprint items should indicate date of entry into the sprint backlog, estimated development time and cost of developing the items. In consultation with the user, the developer uses the prototype created in the requirements stage to create a development plan that enables the delivery of the functionality for the current sprint. Each sprint is also associated with minutes to document agreements between the developer and the user.

Deployment – This stage avails the product or product component to the user, through the execution of the development plan formulated at the Sprint stage. The developed product or product increment is validated with the stakeholders. The validation process is minuted to enable fall back in future. Solo Scrum includes a lot of documentation, mainly inherited from PSP.

Management – This is a cross life cycle activity used to plan for the project execution. It provides for quality reviews at the end of each phase. If sprints are short and equally spaced, then consistency in product delivery is enhanced (Agarwal & Umphress 2008; Pagotto *et al.* 2016; González-Sanabria, Morente-Molinera & Castro-Romero 2017).

The methodologies discussed so far share a number of characteristics, perhaps due to the fact that they are targeted at improving the quality of software and developer productivity in an

industry setting. The case studies to evaluate the utility of most of the methodologies were carried out in industry, although Scrum solo is cited to be in use in an academic setting to develop individual-sized students' software projects. In the following section, DeSoftIn, a methodology specifically designed for use in an academic setting is discussed.

4.3.6 DeSoftIn

DeSoftIn is an agile methodology designed for use by students working on individual software projects in an academic environment (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.25). Derived from existing agile methods it prescribes phases, practices, tools and techniques to be used by students to deliver quality software products. The phases are summarised as follows: -

The phase**,** Planning and analysis initiates the development process. At project onset, users and user roles in the system under development should be identified. Using a checklist that links system functionalities to user roles, the developer captures and prioritises customer requirements on this checklist. These requirements determine project scope. Once the scope is established, the developer carries out a risk analysis for each requirement to determine project feasibility. Requirements are normally identified in advance but may change with project progress. This feature is similar to that in Agarwal and Umphress (2008), where requirements are identified in advance and fixed. If users later request for any changes in these, they are advised to trade in the old requirements for the new. This enables discipline in an academic environment where the project deadline is strict and is set at the beginning of the academic year.

During the Design phase**,** the authors recommend the use of business process model notation, to create high level design of the software so as to incorporate each of the prioritised requirements. The notation facilitates the representation of business processes in a manner that makes it possible for both the user and developer to understand the main processes to be supported by the software (Object Management Group Inc. 2011, p.22). Prototypes may also be developed to help understand complex requirements. DeSoftIn concurs with Scrum solo on the use of a prototype in capturing user requirements.

At the Development phase, the developer creates software code for each functionality, using the prioritised checklist. Programming is done in sprints, so that each functionality is delivered at the end of a sprint. A ten-day sprint is recommended to enable progress tracking. This is

consistent with the recommendation from Bernabé, Navia & García-Peñalvo (2015,p.693), to execute development in sprints lasting for at most two weeks. Colour coding on the checklist can be performed to indicate functionalities outstanding, in progress, under review and approved. A matrix with requirements and user roles is created to log requirements progress against user roles using the colour codes. This is similar to the list in MIDS Adaptation (León-sigg et al. 2018, p.37).

Once the development is complete, Implementation follows. During this phase the developer puts the fully tested software to use. It is recommended that the product be evaluated using quality standards such as ISO/IEC 15504 and ISO 27 000 (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.27). ISO 27000 is a standard that is used for general information systems management (ISO/IEC 2018, p. 1), while ISO/IEC 15504 also known as **S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination (SPICE) is a software process model that defines processes to be evaluated during any software development project to determine the capability of a software process. This is the only methodology that recommends the evaluation of the product using quality standards, particularly for security. However, the authors do not give practices to build security into the product. This research aims to extend this recommendation by proposing practices to be embedded into the methodology in order to promote security. During this phase, the developer also performs risk analysis of the development process. Risk management practices are recommended to handle any identified risks.

Evaluation – At the close of each sprint, the developer meets with the customer to evaluate the work just completed. Since this method is developed for an academic setting, a meeting with the supervisor is also recommended to measure progress so far. The results of the evaluation enable the development team (developer, customer and supervisor) to make adjustments on the items on the checklist, based on current progress.

4.3.7 Initial Software Development Method (MIDS) Adaptation

MIDS Adaptation is developed as a "balanced" software development methodology for use by novice developers (León-sigg *et al.* 2018). The balance seeks to bring about an equilibrium between the agile methods and traditional methods. The original MIDS is designed to support small teams of average size of four persons (León-sigg et al. 2018, p.35). MIDS adaptation is a scaled down version of MIDS that seeks to improve the productivity of solo developers, at

the same time enhancing the quality of their software products. MIDS practices are divided into social, management and development practices. Social practices prescribe how the developer interacts with the users during the development process, and how they capture progress of the development process. Management practices spell out the project management activities to be executed by the developer in a bid to deliver the software product on time, within budget and expected functionality. Development practices spell out the technical activities, tools and techniques for use in each of the stages. This methodology is given special attention in this chapter, since it was not reviewed in Chapter 2 in the meta-synthesis. The social, management and development practices in MIDS Adaptation are summarised in Table 4.1. The tabulation of the practices facilitates an in-depth understanding of the quality practices embedded in this methodology.

Table 4.1: MIDS adaptation practices (adapted from León-sigg et al. 2018)

Adapted MIDS Social, Management, and Development Practices		
Social	**Management**	**Development**
Team Composition -Problem statement & formulation	*Project Planning* -Creation of a software project plan	*Software Requirements* -Use of use case diagrams to document user requirements -Definition of functional & non-functional requirements -Creation of prototypes
Team communication - Definition of team communication and feedback mechanisms	*Iteration Planning* -Use of a simple Kanban board with the columns: To do; Doing; Completed. -Kanban board used for product deliverable scoping.	*Software Design* -Software architecture definition -Software component definition
Creation of personal repository -Definition of documentation standards	*Project Planning & Execution* -Execution of project with the following the Kanban board	*Software Construction* -Software development planning -Creation and testing of code for each user functionality
Project retrospective -Documentation of lessons learnt	*Iteration Assessment and control* -Use of Kanban board to control progress	*Software Integration Tests* -Software Integration -Testing of integrations -Documentation of test results
	Iteration Close -Review of work covered in the iteration	

	-Delivering of iteration product -Review of project repositories	
	Project Close -Delivery of expected product	

As shown in the table, social practices include team composition, team communication, creation of personal repository and project retrospective. Management practices include project planning, iteration planning, project planning and execution, iteration assessment and control and iteration close. Development entails establishing user requirements, software design, software construction, and performing software integration tests. Activities in each of these practices are summarised in the table.

The review of the foregoing methodologies has proved the feasibility (Peffers *et al.* 2008, p.55) of building an SSDM to support product quality in a solo development environment. However, a closer look at the practices in these methods shows that none of the reviewed SSDMs discuss security promoting practices, apart from González-Sanabria, Morente-Molinera and Castro-Romero (2017). The latter limit their discussion to recommending the evaluation of the delivered software product against an appropriate security standard. With this limitation, this research reviewed secure software development literature to identify security practices.

In searching the literature on secure software development, a systematic literature review by Rindell, Hyrynsalmi and Leppänen (2017) was identified. Using the reference section of this publication, more sources discussing secure software development were identified. Section 4.4 below discusses secure software development and identified practices to support software security in the developed software.

4.4 Analysis of secure software development practices.

A number of software security breaches emanate from flaws in the software development process (Ghani, Azham & Jeong 2014; Othmane *et al.* 2014; Mohammad, Alqatawna & Abushariah 2017). Most software development processes and software development organisations do not put the same emphasis on security requirements elicitation as they do on functional requirements (Viega 2005, pp.1-2). As a result, the software development process is inclined towards addressing the functional requirements. Agile methods have excelled in

dealing with the quality of software but not necessarily dealing with the security aspect of software.

On the other hand, research shows that embedding security practices in the software development life cycle improves the security of the resulting software product (Davis 2013; Ghani, Azham & Jeong 2014; Othmane *et al.* 2014). Embedding security practices in the SDLC results in a secure software development life cycle and secure software (McGraw 2005). The Comprehensive, Lightweight Application Security Process (CLASP) (OWASP 2006) is one example process that provides a rich source of practices that can be used to build security into the SDLC. It describes a flexible set of practices that can be applied on demand. Independent developers wishing to build security into their software products can freely access these resources from this pool or from its newer version, the Software Assurance Maturity Model (SAMM)(OWASP 2017).

In this research the aim is to identify lightweight security practices for the purposes of embedding these into the software life cycle to build the proposed methodology. Using the systematic literature review conducted by Rindell, Hyrynsalmi and Leppänen (2017) as a starting point, this section identifies lightweight practices that can be incorporated into the primary SSDM framework derived in Chapter 2. These authors' systematic review was found appropriate for this purpose as it organises identified practices according to the SDLC which corresponds to the primary SSDM. The reference section of these authors was used to identify sources discussing these practices so as to fully understand how they promote security in the developed software. The following sub-sections detail the identified security development practices.

4.4.1 Security standards adoption

Security standards, just like quality standards help to build consistency in the development environment. They help the developer to keep track of the implemented desired security activities during software design. A lone developer may benefit from adopting security standards as those discussed in CLASP (Viega 2005). Example security standards include those for file handling, user authentication, input and output handling and coding and testing standards just to name a few. Standards adopted should be commensurate with the software under development. To enhance productivity, a lone developer should continuously review the available security standards in their line of software applications and create a security

repository of these. Standards reviews can be done in between projects. On undertaking a particular project, these should serve as a baseline for security, and should be updated to meet the current project requirements. Only those standards pertaining to the application at hand need to be considered during a project. To enhance productivity in standards adherence, automated tools may be used.

4.4.2 Conducting security awareness programs

Every developer needs some basic level of training in the development environment. For a secure software development project, training entails acquiring knowledge in secure software development and related practices (Rindell, Hyrynsalmi & Leppänen 2018). Knowledge of secure software development may be obtained through the review of development processes such as CLASP, SAMM (OWASP 2017) and the Microsoft Development Cycle (Microsoft 2008). A solo developer engaging in secure software development may also spend time reading texts such as, 24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them (Howard, LeBlanc & Viega 2010). Here the authors have grouped the twenty-four "sins" into web applications, implementation, cryptographic and networking. Developers wishing to embark on projects with a focus on security, should concentrate on the areas pertinent to their project.

Freely available training manuals and online videos from reputable organisations such OWASP and Microsoft can also be used for training purposes. The developer should seek to acquire basic technical skills such as those for security requirements modelling, secure design, secure coding and secure testing. Such skills enable the developer to handle the multiple roles associated with a solo development environment.

Knowledge acquired on security should be shared with project stakeholders so that they can participate in the identification of threats in their operating environment. User education on security should concentrate on basic security issues pertaining to user roles in the operating environment (OWASP 2017, p.34). Training users on security enables them to actively participate in the identification of misuse cases during the security requirements elicitation process.

4.4.3 Misuse case identification and creation

Misuse cases are an example of threat analysis and modelling tools. Other threat modelling tools include attack trees and the Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges (STRIDE) (Microsoft 2008) approach among others. Threat modelling assists the developer to visualise and keep track of identified security threats so as to design measures to mitigate these. This research concentrates on misuse cases as they are deemed easy to design. This is so, as they mimic use cases used for modelling functional requirements. Identifying misuse cases using abuser stories eases the process, as these can be considered to be the opposite of user stories normally associated with requirements engineering in the agile development approach.

A misuse case portrays a set of unwanted events in a system that cause harm to that system (Sindre & Opdahl 2005, p.34). It represents a hostile actor's actions against a system (Alexander 2003). An actor in this case can be human or any other object that can disturb the smooth operations of a system. Modelling security requirements using misuse cases provides a systematic way of capturing and modelling threats to a system under development. Misuse cases can safely be viewed as use cases from an intruder's point of view of the system. It should be noted that an intruder can launch both a planned or unplanned event. Combining use cases and misuse cases help to communicate security related aspects of the system to stakeholders in an easy to understand way (Alexander 2003; Sindre & Opdahl 2005; OWASP 2017).

End users play a significant part in identifying misuse cases. To help users contribute in the process, the developer can create example misuse cases using known cases in the area of application. This gives the stakeholders examples of what misuse cases are, and encourages users to think widely of what could happen in their environment leading to system unavailability. To simplify the creation process, one can use a top down approach, where one starts by identifying high level threats. As development proceeds, the high level modules can be broken down into their components to identify the finer forms of threats to the system (Alexander 2003).

A systematic process of identifying misuse cases is proposed by Sindre and Opdahl (2005) as:

i. Identify the most important assets of the system (e.g. data, memory or critical processes in the system)

ii. Set goals to secure each identified asset,

iii. Identify threats against each set goal in (ii),

iv. Perform risk analysis on each identified threat,

v. Set goals to mitigate the risks perceived as critical.

To keep the process lightweight, qualitative risk analysis using low, medium, and high (Sindre & Opdahl 2005, p. 36) can be used as this does not need much resources and time. Since it is not possible to deal with all the identified threats, developers usually concentrate on mitigating high risk threats. Properly formulated use and misuse cases serve as a basis for designing security and quality test cases.

Misuse cases can be used to represent the abstract view of system security. This means that some misuse cases that are generic for a number of systems such as illegal login, illegal view of customer details can be reused in future, thus promoting reusability and enhancing productivity. Generic misuse cases like these can be used to form the security repository suggested by Rindell, Hyrynsalmi and Leppänen (2017, p.8). The repository can be continuously refined as the developer discovers new threats and learns how to mitigate these.

4.4.4 Security test definition

Security tests are best defined based on the misuse cases identified during the requirements process. These define a system's attack surface. A system's attack surface is the set of possible threats associated with the system under development (Pressman & Maxim 2015, p.596). Developers start by identifying a system' s attack surface in order to build adequate test cases. All defined tests should be traceable to the threat model used to identify the threats (Maxim & Kessentini 2016, p.30). Possible misuse cases logic paths and expected system behaviour from these should be specified together with associated responses.

Due to resource limitations, solo developers should concentrate on defining test cases for those threats posing high risks on the system. These are those risks ranked as posing as high risk in the threat analysis activity.

4.4.5 Misuse case design

As seen in sub-section 4.4.3, a properly formulated misuse case serves as a basis for both test designs and creating a good system architecture. Design addresses each misuse case logic concentrating on the flow of events to accomplish the misuse case. A good security design

should show how each identified threat in the system is dealt with in the design (OWASP 2017). Models such as sequence diagrams, activity diagrams and class diagrams may be used for the purpose.

In a solo development environment, the design should be simple enough to accommodate future changes. Developers are encouraged to use tools or models they are familiar with. For example, sequence diagrams are used in mainstream software engineering to show the flow of events leading to the fulfilment of a use case. Misuse case sequence can also be modelled similarly showing where the use case is made to fail.

4.4.6 Source code security reviews

Source code security review is an important practice of secure code development. The latter involves the adoption of secure coding standards at the beginning of the software development project. Secure coding standards define practices such as those designed to deal with threats like SQL injection attacks. Examples of practices include user input validation, compiling queries before execution and identifying and avoiding special characters in the input (Palsetia *et al.* 2016, p.95). Security source code review concentrates on high risk modules as modelled using the misuse case diagrams or appropriate threat model (Pressman & Maxim 2015, p.596). Target modules include those receiving data from the outside, interfaces with other systems and access control points (OWASP 2017, p.53).

Solo developers can benefit from automated source code review tools. Automated tools should be used to complement manual reviews for critical points in the system. Tools enhance developer productivity. Trusted open source tools may be used to minimise costs. Code review if automated may be integrated with the development environment as a plugin and set to run at desired intervals.

4.4.7 Security tests

Security testing is a means of establishing that the design and implementation of the system addresses the threats identified during the security requirements stage (Microsoft 2008; OWASP 2017). In test driven development, developers aim to ensure that their code passes all the test cases set. Automation security test tools can be set to run appropriate tests based on the attack surface of the software product (Belk *et al.* 2011).

Various forms of security tests exist, and these are carried out dependent on the threats identified for the software products and the associated impact if that threat happens. In fuzz testing the software product is run with illegal input data to test its behaviour under these conditions. Automated fuzz testing tools can be used where possible to increase productivity as this is normally a lengthy process and would rather conflict with agility, which is key in this thesis. Penetration testing is another form of test that can be used with misuse cases. Penetration testing is defined as a means of simulating an identified attack against a software product (Microsoft 2008). In this case the penetration test seeks to establish whether what has been defined to be a failure point in a misuse case, does certainly fail in the implementation (Belk *et al.* 2011, p.41). A penetration test can be carried out for each critical misuse case identified. Whatever tests are used, the solo developer should opt for lightweight tests.

The in-depth analysis of the reviewed SSDMs and the review of secure software development literature enables the derivation of the quality practices synthesised in Table 4.2 below. The table shows the quality concepts (indicated in the first column), associated quality practices (indicated in the second column) and the quality impact conferred on the developed software product (shown in the third column). The fourth column indicates the authors that discuss the quality concepts identified. Table 4.2 shows that several authors concur on a number of quality practices confirming their effectiveness in software product quality support. These publications and concepts constitute the knowledge base (Hevner *et al.* 2004, p.80; Peffers *et al.* 2019,p .49) from which quality practices are drawn to formulate a higher-quality (Cruzes & Dybå 2011, p. 342) SSDM.

Table 4.2, for example, shows that in the first phase Management Buy-in and Standards adoption, Standards adoption and adherence is the first quality concept. The associated quality practices are adoption of coding standards, adoption of design and documentation standards, user education and adoption of security standards. Adoption of coding standards promotes development consistency and this is a practice drawn from Agarwal and Umphress. Adoption of design and document standards is corroborated by these authors (Agarwal &Umphress 2008; Ramingwong, Ramingwong & Kusalaporn 2017; León-sigg *et al.* 2018). Adoption of security standards is a recommendation from Rindell, Hyrynsalmi and Leppänen (2017) and Viega (2005). Security practices are italicised to distinguish them from quality practices. The rest of the table is interpreted similarly.

Table 4.2: Quality and security promoting practices

Quality Concepts	Quality Practices	Impact on Software Quality	Source
I. Management Buy-in and Standards adoption			
Quality standards	Adoption of coding standards	Maintains code consistency	(Agarwal & Umphress 2008)
	Adoption of design & documentation standards	Standardises design & documentation processes	(Agarwal & Umphress 2008; Ramingwong et.al. 2017; León-sigg et al. 2018)
	Adoption of security standards	*Enhances product security*	*(Viega 2005; Rindell, Hyrynsalmi & Leppänen 2017)*
Education	Educating users on methodology	Prepares users to participate in development; Enhances user acceptance	(González-Sanabria, Morente-Molinera & Castro-Romero 2017; Ramingwong, Ramingwong & Kusalaporn 2017)
	Institution of security awareness programs	*Enables users to participate in identifying misuse cases*	*(Microsoft 2008; OWASP 2017; Rindell, Hyrynsalmi & Leppänen 2017)*
II. Requirements Elicitation			
User requirements identification	Creation of user stories using the INVEST acronym	Facilitates approximate time estimation	(Bernabé, Navia & García-Peñalvo 2015)
	Keeping user stories simple enough	Enhances requirements understandability	(Agarwal & Umphress, 2008; Bernabé, Navia & García-Peñalvo 2015)
	Use of simple metaphors	-Enhances requirements understandability -Enhance product testability	(Agarwal & Umphress 2008)
	Use of small story cards to capture requirements	Simplifies user requirements	(Ramingwong, Ramingwong & Kusalaporn 2017)

	Creation of a requirements checklist linked to system roles	Identifies and addresses all user requirements	(González-Sanabria, Morente-Molinera & Castro-Romero 2017)
	Creation of Use cases		(Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; León-sigg *et al.* 2018)
	Creation of misuse cases for each use case	*Enables focus on countering security threats*.	(*Alexander 2003; Sindre & Opdahl 2005; Rindell, Hyrynsalmi & Leppänen 2017)*
	Creation of prototypes	Clarifies user requirements	(Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; González-Sanabria, Morente-Molinera & Castro-Romero 2017)
	Creation of product backlog	Captures and prioritises user requirements; Controls development status	(Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; González-Sanabria, Morente-Molinera & Castro-Romero 2017)
Scope Definition	Use of epic stories	Abstracts product functionality	Bernabé, Navia & García-Peñalvo 2015)
	Creation of Work breakdown structure (WBS)	Captures all work to be done	(Pagotto *et al.* 2016)
	Creation of Product breakdown structure (PBS)	Shows all product components	(Pagotto *et al.* 2016)
III. Release and Sprint Planning			
Development productivity	Definition of sprint objective	Keeps developer and user focused	(Bernabé, Navia & García-Peñalvo 2015)
	Development of unit tests	Focuses development effort	(Agarwal & Umphress 2008; Pagotto *et al.* 2016; González-
	Use of short sprints (3 to 14 days)		

			Sanabria, Morente-Molinera & Castro-Romero 2017)
	Risk Analysis	Reduces negative impact on identified risks	(González-Sanabria, Morente-Molinera & Castro-Romero 2017)
Enhancing security	Definition of security acceptance tests for each use case	Prepares user and developer for product delivery	(Rindell, Hyrynsalmi & Leppänen 2017)
	Design of misuse cases	*Builds security into the rest of the system architecture*	*(Alexander 2003; Sindre & Opdahl 2005; Rindell, Hyrynsalmi &Leppänen 2017)*
IV. Development with Review			
Development transparency	Development time estimation in hours	Speeds up development progress	(González-Sanabria, Morente-Molinera & Castro-Romero 2017)
	Prioritised product backlog	Ensures Product completeness	(Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; Ramingwong, Ramingwong & Kusalaporn 2017)
	Prioritised sprint backlog	Requirements addressed according to the user's priority	(Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016; Ramingwong, Ramingwong & Kusalaporn 2017)
	Use of equally spaced milestones	Deliver components regularly	(Bernabé, Navia & García-Peñalvo 2015)
	Burndown charts	Visualise progress	(Ramingwong, Ramingwong & Kusalaporn 2017)
	Coded/colour Kanban board/digital dashboards/logbook/logfile	Visualise product backlog progress	(Dzhurov, Krasteva & Ilieva 2009; González-Sanabria, Morente-Molinera

			& Castro-Romero 2017; León-sigg et al. 2018)
Ensuring Code Quality	Explanation of code to dummy partner/ Self-dialogue	Reduces code defects	(Agarwal &Umphress 2008; Bernabé, Navia & García-Peñalvo 2015)
	Unit testing		(Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009; León-sigg et al. 2018)
	Performing of source code level security reviews	*Identifies security flaws in code*	*(Palsetia et al. 2016; OWASP 2017; Rindell, Hyrynsalmi & Leppänen 2017)*
Enhancing development productivity	Task automation	Reduces time taken to implement task	(Bernabé, Navia & García-Peñalvo 2015)
	Automated code review	Enhances defect identification	(Dzhurov, Krasteva & Ilieva 2009; Pagotto et al. 2016)
Simplifying product design	Use of CRC cards	Shows class relationships	(González-Sanabria, Morente-Molinera & Castro-Romero 2017)
	Creation of simple product architecture	Focus on core product functionality	(Dzhurov, Krasteva & Ilieva 2009; León-sigg et al. 2018)
V. Sprint Review and Close			
Ensuring code quality	Code Refactoring	Reduces risk of defects	(Agarwal &Umphress 2008)
		Improved code quality	(Agarwal &Umphress 2008; Dzhurov, Krasteva & Ilieva 2009; Bernabé, Navia & García-Peñalvo 2015)

	Use of test suites to implement test driven development	Speeds up unit testing	(Agarwal & Umphress 2008)
	Use of version control systems	Enhances product maintainability	(Agarwal & Umphress 2008; Bernabé, Navia & García-Peñalvo 2015)
	Performance of code walkthroughs	Reduce code defects	(Agarwal & Umphress 2008)
	Code coverage tests	Reduces code defects	(Dzhurov, Krasteva & Ilieva 2009)
	Implementing security tests	*Reduces security flaws in code*	*(Maxim & Kessentini 2016; Rindell, Hyrynsalmi & Leppänen 2017)*
	Continuous code integration	Reduces deviation from main code base	(Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009; Bernabé et al. 2015; González-Sanabria, Morente-Molinera & Castro-Romero 2017)
	Performance of integration test	Reduces system defects	(Agarwal & Umphress 2008; Dzhurov, Krasteva & Ilieva 2009; Bernabé, Navia & García-Peñalvo 2015; González-Sanabria, Morente-Molinera & Castro-Romero.2017)
	Frequent sprint breaks	Reduces developer burnout	(González-Sanabria, Morente-Molinera & Castro-Romero 2017)
VI. Evaluation			

Acceptance	Use of Acceptance tests	Enhance user acceptance	(Agarwal & Umphress 2008; Bernabé, Navia & García-Peñalvo 2015)
Communication	Use of a system metaphor	Enhances requirements understanding	(Agarwal & Umphress 2008)
	Use of acceptance register	Enhances user participation	(León-sigg et al. 2018)
Delivery frequency	Continuous delivery	-Allows for developmental control -Enhances user participation	(Agarwal & Umphress 2008; Bernabé, Navia & García-Peñalvo 2015; González-Sanabria, Morente-Molinera & Castro-Romero 2017; León-sigg et al. 2018)
	Use of small milestones	Ensure frequent component delivery	(Agarwal & Umphress 2008; González-Sanabria, Morente-Molinera & Castro-Romero 2017; León-sigg et al. 2018)

Table 4.2 has shown that there are practices in the existing literature that can be used to promote both quality and security in a solo development environment. Section 4.5 presents the requirements of the Secure-SSDM.

4.5 Secure-SSDM Requirements

As specified in the aim, and to encourage its uptake, the Secure-SSDM is designed to be lightweight. The lightweight characteristics of the methodology were drawn from the agile manifesto and existing agile solo software development methods reviewed at the beginning of this chapter. Only those principles from the manifesto that apply to a solo environment were deemed important.

4.5.1 Lightweight methodology

The following agile principles were deemed important for the Secure-SSDM:

i. Satisfy the customer through early product delivery

This is a risk mitigatory measure. As the solo developer delivers the product in prioritised increments, those product components that the customer considers of high priority are delivered first. This gives the customer the opportunity to test and accept the components as they are being developed without having to wait for the whole product at the end of the project. Secure-SSDM should support early and incremental delivery of software.

ii. Incorporate requirements change throughout the development process

Incorporating changes throughout the development cycle ensures that the developer keeps pace with the user's preferences during the course of the project. This is a principle that guards against the delivery of a product that no longer serves the desired purpose. However, to avoid scope leap, change should be controlled and developers should use version control systems to track changes. Further, developers should weigh and make decisions on whether or not changes should be implemented at any point in time (Bernabé, Navia & García-Peñalvo 2015, p.688).

iii. Deliver working software frequently, preferably in short cycles

This principle is linked to that of satisfying the customer early in the development cycle. The methodology should facilitate timely delivery of working software to the customer, preferably in increments. The recommended incremental durations from the reviewed solo software development publications is two to four weeks (Bernabé, Navia & García-Peñalvo 2015; Pagotto *et al.* 2016).

iv. Continuous customer involvement

In solo development environments, the quality of the software rests on the developer and their interaction with the user. User involvement in the development process enhances software product acceptability by the customer. Practices that support continuous customer involvement should be evident from the methodology.

v. Measure project progress using working software

For a solo developer this practice gives them impetus to continue with the development as they see tangible results at the user's site. This also helps to gauge the required time and

resources to complete the project (González-Sanabria, Morente-Molinera & Castro-Romero 2017). Further, measuring progress through working software gives assurance to the user that the developer has capacity to deliver as promised. Measuring progress here entails evaluating how much functionality has been delivered together at the same time testing whether the delivered components meet expected security requirements.

vi. Uphold a sustainable development process

The methodology should enable maximum discipline in the developers. It should facilitate tracking of developer progress (Ramingwong, Ramingwong & Kusalaporn 2017, p. 345). Further, it should ease the measuring of developer's speed of progress enabling the computation of outstanding project work and the time required to complete the work (Amjad et al. 2017, p.5825). Provision for visualising progress is key to a sustainable development process (Amjad et al. 2017, p. 5825; León-sigg et al. 2018, p. 38). This was also established to be a key requirement from students that participated in the academic case study to evaluate the Secure-SSDM.

vi. Focus on technical and design excellence

Product quality is heavily dependent on design excellence, which is a core concept of agility (Doyle *et al.* 2014). Design excellence in agile methods like XP is achieved through making the design simple enough to allow for change in case it is required in future (Fioravanti 2011).

vii. Ensure maximum simplicity

Keeping the design simple from a solo development perspective ensures that the developer does not waste time on complex designs that may not deliver expected functionality. The two PXPs reviewed in this thesis advocate for design simplicity. The Secure-SSDM should enable the production of simple designs, simple enough so that modules can be tested independently, easy to understand, supporting ease of navigation to locate desired components, and easily understood by other developers (Pagotto *et al.* 2016, p.687). As an independent developer there may be no other developer to understand your code, but if there is need to maintain your code in future, simplicity makes the maintenance process much easier.

viii. Allow developers to reflect on performance and adjust processes accordingly.

Secure-SSDM is designed for solo developers, therefore support for adjustment of their processes is very important. Process adjustment is also upheld by some of the authors reviewed in this chapter(Dzhurov, Krasteva & Ilieva 2009; Pagotto *et al.* 2016).

4.5.2 High Quality

Software process quality is the ability of a software development methodology to produce high-quality software products (García-Mireles et al. 2015, p.150). As defined in the ISO/IEC 25010 quality model, a software product is of high quality if it displays: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability (ISO 2010). These are high level characteristics with sub-characteristics. The sub-characteristics at the concrete level of this model provide a means for measuring high level characteristics. Sub-characteristics provide a measure for one or more characteristics at the high level. A comparison of the quality characteristics derived from existing solo software development methodologies (SSDMs) with this model showed that activities in the derived framework fully support maintainability and functional suitability which are abstract characteristics. Characteristics such as usability and reliability were seen to be partially supported. Performance efficiency, compatibility, security and portability were not supported. This research proposes to close this gap by incorporating security promoting practices in the primary SSDM. The resulting methodology should therefore provide support for usability, reliability, maintainability, functional suitability and security.

The main knowledge contributions in this research can be summarised as: -

i. The design and evaluation of a Secure-SSDM with quality practices that build quality into the resulting software products, and

ii. The addition of security promoting practices into the solo software development body of knowledge.

The following paragraphs discuss the characteristics required of the methodology to enable the building of the quality into the resulting software product.

Support for Product Maintainability

Product maintainability refers to the ease with which a software product can be adapted to address changes in the environment (Nistala et al. 2016, p. 138). From the existing methods, the research established that test driven development, refactoring and unit testing enhance

code quality at the same time promoting product maintainability (Dzhurov, Krasteva & Ilieva 2009¸ p. 258 ; Bernabé, Navia & García-Peñalvo 2015, p. 690). Similarly, the use of version control systems during software development was seen to enhance product maintainability (Bernabé, Navia & García-Peñalvo 2015, p.690). A version control software like Git can be helpful to a solo developer as it enables the developer to track changes in their code (Driessen 2010; Bernabé, Navia & García-Peñalvo 2015). Driessen (2018) discusses a set of tools in Git that can be used by a developer to keep track of changes. These include tools for accessing recently modified code, making corrections on erroneously committed code and making amends on committed code.

Support for Product Functional Suitability

Functional suitability is a measure of how far a delivered product meets user requirements. This can also be viewed as product functionality. It is defined as "the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions" (Nistala *et al.* 2016, p.138). Functional suitability of a software product is determined by completeness, correctness and appropriateness at the concrete level (ISO 2010). Completeness refers to the extent to which users' objectives have been met by the product, correctness measures the exactness of the expected results, while appropriateness gives a measure of how the delivered product is able to support the tasks at hand. The derived framework supports completeness and correctness.

From the derived framework, the use of a product backlog during requirements elicitation (Bernabé, Navia & García-Peñalvo 2015, p.689) and the use of a work breakdown structure during release and sprint planning was touted to promote product completeness (Dzhurov, Krasteva & Ilieva 2009). A product backlog is a set of features expected by the user from a software product. This is normally created at project start. A work breakdown structure created from the product backlog helps the developer to get a full understanding of the customer product.

Support for Product Usability

Usability measures the utility of the software to the intended user. It is defined as "the extent to which a product can be used by specific users to achieve specific goals with effectiveness, efficiency and satisfaction in a specific context of use" (ISO 2010; Nistala et al. 2016, p. 138).

The quality theory derived from the reviewed SSDMs shows that usability is promoted by simplicity and understandability at the concrete level. In turn, simplicity was shown to be promoted by the adoption of development standards, use of small user stories and small tasks, practicing automated code reviews, refactoring large user stories and components and the production of testable code. These practices should therefore be incorporated in the proposed methodology.

Support for Product Security

Product security is defined as the degree to which a product or system protects information and data so that persons or other products or systems are afforded the degree of data access appropriate to their types and levels of authorisation (ISO 2010). As defined in the ISO/IEC 25010 Quality model, product security has the following sub characteristics: confidentiality, integrity, non-repudiation, accountability and authenticity. These sub-characteristics deserve special attention as they constitute a major contribution in this thesis.

Confidentiality measures the degree to which data access is restricted to authorised users (ISO/IEC 2018, p.2). This is an important functionality for software products handling business and personal data. Software products handling customer details and data should ensure that these are only accessed by those users with access rights. Integrity measures the degree to which a software product prevents unauthorised changes to data and information as a means of maintaining data accuracy (ISO/IEC 2018, p.5). A software development methodology seeking to promote data integrity should incorporate practices that restrict access and modifications to data to authorised users only. Data should always hold the meaning it was originally meant to convey. Non- repudiation pertains to the capability of a system to notice the occurrence of activities performed against the data and the system (ISO/IEC 2018, p.6), at the same time tying users to their actions. Accountability gauges the ability of the system to successfully identify the user who accesses a system component, so that they have no room to deny the act (ISO 2010). Authenticity pertains to the assurance that the object claiming access to data or parts of a system is what it says it is (ISO/IEC 2018, p.2).

Security promoting practices are the main contribution in this research. This research concurs with Maxim and Kessentini (2016, p. 29) that security should concern all those developers seeking to deliver quality software. Clients of web-based systems in particular require maximum security on their websites (Haq *et al.* 2018). At the evaluation stage of this research, the Secure-SSDM is used in a multiple case study to design and develop web-based

applications so as to evaluate its utility (Hevner *et al.* 2004) in building secure software products.

Whereas focusing on product security has been viewed as contradicting agility, several researchers have explored the concept of incorporating security promoting practices into agile methods without necessarily compromising the agility of the resultant methodology. Rindell, Hyrynsalmi and Leppänen (2017) refuted the contradiction between agile practices and security practices. They did this through outlining a set of secure agile practices that cover the software development cycle, drawn from the extant literature. This has been elaborated in Sub-section 4.4 where an analysis of the suggested security practices is detailed. Pohl and Hof (2015) confirm the refutation through the development and evaluation of a secure version of Scrum, which they called Secure-Scrum.

The foregoing paragraphs detail the requirements to be satisfied by the Secure-SSDM. The requirements were deemed essential based on the reviewed literature. Further requirements were collected from the developers who applied the methodology in designing application products. This was done through eliciting their perceptions on the utility of the methodology, as well as suggestions for improvement. Using DSR (Peffers *et al.* 2008) these requirements serve as an evaluation benchmark to test the developed artefact at the end of the research. The next chapter, Chapter 5 discusses the design and implementation of the Secure-SSDM.

4.6 Chapter Summary

This chapter presented an in-depth analysis of the quality promoting practices in existing SSDMs. A further analysis of lightweight security practices was performed. Following the DSR (Peffers *et al.* 2008, p. 55), this chapter served to define the objectives to be fulfilled by the Secure-SSDM in enhancing the quality of software products designed by solo developers. Quality practices as defined by the authors of the reviewed literature in solo software development literature were extracted, together with the authors' views on how they produce the desired impacts on the resulting product. A case study with undergraduate students at the National University of Science and Technology (NUST), Zimbabwe was used to refine the artefact objectives so as to address the peculiar needs of solo developers. A similar case study conducted with three developers from industry served to further perfect the requirements.

The extant literature on SSDM and associated literature on secure software development, aids the classification of the Secure-SSDM objectives into two broad categories. The two

categories are methodology agility, and that for support in delivering high-quality products. To be deemed agile, the Secure-SSDM should facilitate: satisfaction of the customer through early product delivery; incorporation of requirements change throughout the development process; frequent delivery of working software, preferably in short cycles; continuous customer involvement; measuring of project progress using working software; upholding of a sustainable development process; focus on technical and design excellence as well as ensuring maximum simplicity. These are agile principles drawn from the Agile manifesto (Beck et al. 2001; Fowler & Highsmith 2001).

On the other hand, support for quality is demonstrated by support for product maintainability, usability, functional suitability and security. Chapter 5 presents a detailed design of the Secure-SSDM and demonstrates how design rigour (Hevner *et al.* 2004, p.84) was applied in building the artefact.

CHAPTER 5 SECURE-SSDM DESIGN

5.1 Introduction

In Chapter 4 an in-depth analysis of the solo software development (SSD) environment was conducted enabling the derivation of requirements for the secure solo software development methodology (Secure-SSDM). The analysis of practices derived from the existing methods formed a basis for the methodology requirements. Further, the investigation of security promoting practices from existing secure software development methodologies served to complete the high-level methodology requirements. The high-level requirements for the methodology under design are that it should support software development agility, at the same time promoting quality in the developed software. To support agility the Secure-SSDM should promote: satisfaction of the customer through early product delivery; incorporation of requirements change throughout the development process; frequent delivery of working software, preferably in short cycles; continuous customer involvement; measuring of project progress using working software; upholding of a sustainable development process; and focus on technical and design excellence while promoting maximum simplicity. Section 4.4.1 of Chapter 4 explains these agile concepts. To promote the delivery of quality software products, the methodology should enable maintainability, usability, functional suitability and security in the developed software.

In this chapter, a befitting design towards fulfilling the enlisted requirements is discussed. Chapter 5 provides an answer to the fourth research question posed in this thesis thus:

SQ4. How can quality and security practices from lightweight software development methodologies be synthesised into a solo software development methodology that promotes quality and security in the developed software?

The Secure-SSDM is designed iteratively following the Design science research (DSR) cycle of : (1) Problem identification; (2) Definition of solution objectives; (3) Design and development; (4) Solution demonstration; (5) Solution evaluation; and (6) Results communication (Peffers *et al.* 2008, p. 53). The iteration during the design phase achieves the rigour necessary for DSR projects. Sections 5.2 and 5.3 of this chapter discuss the design and development of the methodology artefact, after the problem identification and objectives formulation were dealt with in Chapter 2 and Chapter 4 respectively. The utility of the resulting artefact from the first design iteration was demonstrated through soliciting for

criticism and feedback from participants at a Computing research seminar. This was followed by the presentation of the primary SSDM framework at a peer-reviewed international conference. Comments on the utility of the framework were used to refine the artefact. A case study with undergraduate students studying towards a Bachelor of Science Honours Degree in Computer Science served to further demonstrate and evaluate the utility of the methodology. The student participants were asked to use the Secure-SSDM to develop individual sized projects. This demonstration of the utility of the methodology is detailed in section 5.4, and serves to prove that the Secure-SSDM can be used to develop high-quality and secure software products (Peffers *et al.* 2008, p.55). Feedback obtained from the students after the case study was used to further refine the artefact. To deal with the case of external validity, industry developers were solicited to further prove the artefact's utility in an industry case study. Three industry developers, each with a minimum qualification of a degree in Computing (Computer Science & Information Technology) and an average of four years software development experience participated in the case study. These two case studies constitute the solution demonstration and evaluation stages of the DSR. The evaluation part and its results are discussed in Chapter 6.

5.2 Secure-SSDM Design

Design is a wicked problem, particularly in software engineering where the process involves the building of complex artefacts with human and technical components whose functional and quality characteristics are inseparable (Baskerville et al. 2018, p.362). The artefact under design in this research is an agile software development methodology. The methodology is designed for use by solo (freelance) developers in building quality and security into their software products. As indicated in Section 5.1, the Secure-SSDM is designed to embed quality and security promoting practices in its life cycle stages. The assumption here is that, the embedded security and quality practices promote the development of high-quality and secure software products. This section shows how the security promoting practices are integrated with the methodology's core quality promoting practices to give the methodology's expected properties. Selected security practices from the agile security framework of Rindell, Hyrynsalmi and Leppänen (2017) and the reviewed related security literature are integrated into the six-stage SSDM framework derived from the literature in Chapter 2 to produce the Secure-SSDM. The chosen security framework was found appropriate as its security practices are organised into six stages of the SDLC which neatly fit into the six stages of the primary

SSDM. It was therefore possible to identify practices appropriate for each SSDM stage. Only those security practices that could be executed by an individual were identified and incorporated into the development stages, taking care not to compromise the agility of the methodology. At this stage, the following specific design related question is posed:

How can existing secure software development practices be integrated into the SSDM framework to build a secure SSDM without compromising the resulting methodology's agility?

To answer this design question, literature discussing the integration of security practices into agile software development processes was reviewed. Section 5.2.1 discusses the reviewed literature and the method that is subsequently defined for the integration process.

5.2.1 Embedding security practices into Agile methods

Embedding security promoting practices into software development methods is a cost cutting measure as this eliminates the need for an external security resource. Such a move enhances software quality, at the same time promoting the production of secure software products (Sonia & Singhal 2012). However, embedding secure software development practices into agile methods is not an easy task (Keramati & Mirian-Hosseinabadi 2008; Sonia & Singhal 2012; Sonia et al. 2014; Oueslati et al. 2015; Rindell et al. 2018). Adding available security promoting practices to agile methods may compromise the agility of the resultant method if appropriate measures are not taken. A need therefore arises to methodically integrate security practices into agile methods without reducing the agile characteristics of the final artefact.

Several researchers (Beznosov & Kruchten 2004; Keramati & Mirian-Hosseinabadi 2008; Sonia & Singhal 2012; Sonia et al. 2014; Rindell et al.2018) have tackled the problem of introducing security practices into lightweight methods. Beznosov and Kruchten analysed the compatibility of traditional security promoting practices with agile methods. They produced a list of compatible, independent, partially automatable and mismatch practices. Compatible and independent security practices could readily be integrated with existing agile practices. The problem was dealing with partially automatable and mismatch practices. They recommended automation supported by knowledge management for partially automatable practices, and either designing new agile compatible security practices or applying traditional security practices, at least two times within the agile development process, for the mismatch lot (Beznosov & Kruchten 2004, p.51). The authors concluded by posing a question on how

to seamlessly integrate security practices into the agile development environment without compromising the agility of the resulting method.

Progressing knowledge in this area of research, Keramati and Mirian-Hosseinabadi (2008) designed an algorithm for the identification and integration of security practices with existing agile practices, taking care to maintain agility in the resulting practices. The algorithm computes an agility degree for the identified security practice, after which the practice is integrated if and only if, it meets a certain agility threshold. In this case, the project team, or organisation wishing to introduce security practices into its agile environment, determines the agility threshold. This approach to threshold determination works in a project specific environment or organisational setting, but may not be suitable for a generic environment such as the one for this research where the aim is to design a secure methodology for use in any project environment. Nevertheless, the idea of a set threshold may be useful in controlling the integration process.

Sonia, Singhal and Banati (2014) designed Fisa-XP, a secure agile framework. This is a security practices integration framework designed through combining XP practices with secure software development practices drawn from Open Web Application Security Project (OWASP)'s Comprehensive Lightweight Application Security Process (CLASP). The researchers use a modified version of Keramati and Mirian-Hosseinabadi (2008)'s algorithm to identify appropriate security practices from CLASP which they integrate with XP practices. An automated tool, Tisa-XP, is used to compute the agility degrees of identified security practices. This automated tool helps to ease the integration process. Further, the tool is designed to provide guidance on how to implement the security practices recommended for integration with agile practices. The inbuilt tutorial on practice execution is most applicable in solo environments, where developers may not have the necessary security expertise and have no one to consult for assistance.

The examples above, show how researchers have separately, either proved the possibility of integrating security practices with agile methods or demonstrated their successful integration at the same time maintaining the agility of the resulting process or methodology. In this research, Keramati and Mirian-Hosseinabadi (2008)'s algorithm is adapted to identify a list of security practices from Rindell, Hyrynsalmi and Leppänen (2017)' s security development framework and related literature. Only those practices that can be performed by an individual were identified for integration with the SSDM framework designed in Chapter 2 to produce

the Secure-SSDM. In the following subsection, the identification and integration processes are discussed.

5.2.2 Integrating quality and security practices

Keramati and Mirian-Hosseinabadi's algorithm is most suitable for an organisational setting, where a security team scans the environment for security practices that can possibly be integrated with agile practices within that organisation. The algorithm works with a list of agile practices and a list of security practices, both with independently computed agility degrees. Keramati and Mirian-Hosseinabadi (2008)'s algorithm can be summarised using the following steps:

1) Select a security practice with the highest agility degree from the security practices list.

2) Scan the list of agile practices to be integrated with security practices to identify all those that can be integrated with that practice. Choose the one with the least agility degree for integration, if none exists, delete the security practice from the list and stop (go to 6). *There is need to create an agile and security practices compatibility matrix in order to execute this stage*.

3) Generate a new secure agile activity through integrating the agile activity and the security activity, compute its agility degree as min (a, b) where a is the agility degree of the agile practice and b, the agility degree of the security practice.

4) Check if original agility degree of the agile practice + ART >= new secure agile activity's agility degree and integrate the two, otherwise integration is deemed impossible.

5) Remove security practice from the list.

6) Stop or go back to 1 if security practices still exist.

The main adaptation in this algorithm is in step 4, on the threshold value. In that step, ART is the agility reduction threshold meant to control the integration of the two practices, and is based on the project team's capabilities to absorb the security practice, as well as the organisational practices and culture. In this research the ART parameter is inapplicable, therefore this is adapted so that only activities with a resulting agility degree >= 0.5 after

integration are integrated. This is in line with the recommendation by Qumer and Henderson-Sellers (2008, p. 281) to consider any practice or methodology with an agility degree >= 0.5, as agile.

In this case, the research therefore adopts the agility values of 0 to 1 as suggested by these authors. This is important for the nature of the methodology under development. Since the methodology under development is generic, the use of a generic value is most appropriate. Before the adapted algorithm can be applied to the design process, there is need to identify the core development practices of the SSDM framework. Those practices with high occurrences (confirmed by three or more authors) among the SSDMs participating in the meta-synthesis of Chapter 2, were chosen. Thus, the Secure-SSDM is built on development practices generally accepted in the SSDM community (Peffers *et al.* 2008, p.52). The rest of the practices become optional practices which are executed on demand, depending on the type of product under development. It should be noted that the Secure-SSDM does not restrict developers from adopting any agile practice in a bid to improve the quality of the product. Developers are encouraged to practice good knowledge management so that they can keep those practices that work for their environments, and replace those that do not with new ones that do. Table 5.1 shows the core practices of the framework obtained through publication consensus. SSDM core practices and security practices can only be integrated if they are compatible (Keramati & Mirian-Hosseinabadi 2008; Sonia et al. 2014). Two practices are compatible if the developer can execute the two simultaneously with minimal effort. A compatibility matrix (Table 5.3) derived mainly from the literature and close analysis of the practices, was created for the purpose.

In computing degrees of agility of the core practices, this research adopts Qumer and Henderson-Sellers (2006b, p.505)'s definition of agility thus: "Agility is a persistent behaviour or ability of a sensitive entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span, uses economical, simple and quality instruments in a dynamic environment and applies updated prior knowledge and experience to learn from the internal and external environment". From this definition, these authors further define five agility features, which are flexibility, swiftness, leanness, responsiveness and learning. These five features are then used to derive the agility degree of an object. In this research a sixth feature, simplicity, was added to the five features as it was deemed important for the Secure-SSDM under development. These six features were used to

compute the agility degree of each of the SSDM's development core practice as suggested in Keramati & Mirian-Hosseinabadi (2008). Each core practice was analysed to check whether it exhibits the six agility features. The presence of a feature is signified by a 1 (present), absence by a 0 (not present). These contribute to the agility values of the practice, so that a practice exhibiting all the six features has an agility degree of 6/6 = 1. In computing the agility values of the agile practices in the SSDM, reference was also made to the works of Qumer and Henderson-Sellers (2006; 2006b) where the agility values of Scrum and XP are computed. This research is similar to these authors' in that most of the SSDM practices were drawn from existing SSDMs which in turn draw their practices from XP and Scrum. An example illustrating the computation of the agility degree of the User identification development practice is explained in the next paragraph.

First, there is need to check whether the practice exhibits any of the agility features, where the existence of a feature is signified by a 1 and the inexistence by a 0. User identification in the SSDM framework is a flexible process. Users can be added and removed from the process depending on their needs, therefore a 1 is assigned for this feature. User identification can also be done quickly, resulting in another 1 being assigned for speed, although it involves some documentation, hence it is not a lean process. A 0 is assigned for leanness. This is a flexible process, (a 1 is assigned for flexibility), since users can be added and removed as per customer's need, hence it is also a responsive practice, and a 1 is assigned for responsiveness. Lastly, this is a simple process and in turn simplifies the development process, therefore, a 1 is assigned for simplicity. This information is illustrated in the first row of Table 5.1. The values assigned to this activity when summed up, over the total possible sum give: 5/6 = 0.83. The degree of agility for this practice is therefore 0.83. The rest of the degrees of agility for each of the practices were computed in a similar manner and are given in Table 5.1.

Table 5.1: Computing SSDM core practices degrees of agility

Feature / SSDM practice	Flexibility	Speed	Leanness	Learning	Responsive-ness	Simplicity	Degree of agility
I. Management Buy-in and Standards Adoption							
User identification	1	1	0	1	1	1	5/6 = 0.83

User education	1	0	0	1	1	1	4/6 = 0.67
Standards Adoption	1	1	0	1	1	1	5/6 = 0.83
High-level user requirements identification	1	1	0	1	1	1	5/6= 0.83
II. Requirements Elicitation							
Prioritisation of product backlog	1	1	0	1	1	1	5/6=0.83
Prototype development	1	1	0	1	1	0	4/6 = 0.67
III. Release and Sprint Planning							
Creation of user stories	1	1	0	1	1	1	5/6 = 0.83
Prioritisation of Sprint tasks	1	1	0	1	1	1	5/6= 0.83
Design of acceptance tests	1	1	0	1	1	0	4/6 = 0.67
IV. Development with Review							
Coding	1	1	0	1	1	1	5/6 = 0.83
Version/change control tracking	1	1	0	1	1	0	4/6 = 0.67
Code refactoring	1	1	1	1	1	0	5/6 = 0.83
Code review with dummy	1	0	0	1	1	1	4/6 = 0.67
Product validation	1	1	1	1	0	1	5/6 = 0.83
V. Sprint Review and Close							
Sprint review	1	1	0	1	1	1	5/6 =0.67
Project progress review	1	1	0	1	1	1	5/6 = 0.67
Continuous code integration & testing	1	1	1	1	1	0	5/6 = 0.83
Next Sprint planning	1	1	1	1	0	1	5/6 = 0.83
VI. Evaluation							
Deliverables evaluation	1	1	1	1	0	0	4/6 = 0.67
System testing	1	1	1	1	1	0	5/6 = 0.83
Task automation	1	1	1	1	1	1	6/6 = 1

Using the adapted algorithm, with the agility degrees of the core development practices at hand, the next thing is to determine the agility degrees of the security practices. Table 5.2 shows the selected security practices drawn from Rindell, Hyrynsalmi and Leppänen (2017) and their computed degrees of agility based on the same approach used for the SSDM quality

practices. In both cases the agility degree of any practice depends on the experience and expertise of the developer executing that practice. An experienced developer may execute a given practice faster than a novice, and similarly find a practice simpler as compared to a novice. In this case the research assumes average developer experience.

Table 5.2: Computing agility degrees of security practices

Feature / Security Practice	Flexibility	Speed	Leanness	Learning	Responsive-ness	Simplicity	Degree of agility
Security awareness training	1	0	0	1	1	1	4/6 = 0.67
Security analysis of user roles	1	0	0	1	1	0	3/6 = 0.5
Misuse case detailing	1	0	0	1	1	1	4/6 = 0.67
Application of Security design principles	1	0	0	1	1	0	3/6= 0.5
Security test design	1	1	0	1	1	0	4/6=0.67
Security coding standard adherence	1	1	0	1	1	0	4/6 = 0.67
Source code security reviews	1	1	0	1	1	1	5/6 = 0.83
Security testing	1	1	0	1	1	0	6/6= 0.67
Security disclosure management	1	1	0	1	1	0	4/6 = 0.67
Review of security repository	1	1	0	1	1	1	5/6 = 0.83

Once the degrees of agility have been determined for the two groups of practices, the next step is to create a compatibility matrix indicating compatible and non-compatible practices between these practices. Table 5.3 is a compatibility matrix showing the compatibility

between security practices and the primary SSDM quality practices. Each row shows an SSDM practice, and each column shows a security practice. For each SSDM practice, there is need to check its compatibility with all the identified security practices. Practices are compatible if they appear in the same stage of the software development cycle (Sonia & Singhal 2012), or if they can be simultaneously executed with minimal reduction of developer productivity (Keramati & Mirian-Hosseinabadi 2008). To compile the compatibility matrix the quality practices and the security practices were independently organised into the broad stages of the SDLC which are requirements analysis, design, development and testing. Security practices in the requirements analysis stage are deemed compatible with quality practices in that stage. Reference was also made to the works of Beznosov and Kruchten (2004), Keramati and Mirian-Hosseinabadi (2008), Sonia and Singhal 2012 as well as Rindell et al. (2018) in determining the compatibility between practices.

To illustrate the creation of the compatibility matrix, the first quality practice in Table 5.3 is used. User identification is a requirements analysis practice carried out in the early stages of the cycle. The table shows that user identification is not compatible (NC) with misuse case detailing, application of security design principles, security test design, source code security reviews, security testing and security disclosure management. This practice is however shown to be compatible (C) with security awareness training, security analysis of user roles, and review of security repository. The understanding is that while the developer is identifying system users, they may also carry out security analysis on the kind of activities the users play on the system, conduct security awareness training, and at the same time if there is an already existing security repository, they may update it based on the roles users play on the system. The rest of the entries in the table were arrived at using the same logic. However, to keep the practices as agile as possible, only one most favourable security practice is combined with one SSDM core practice.

Table 5.3: SSDM and security practices compatibility matrix

Security practice / Quality practice	Security awareness training	Security analysis of user roles	Misuse case detailing	Application of security design principles	Security test design	Security coding standard adherence	Source code security reviews	Security testing	Security disclosure management	Review of security repository
User identification	C	C	NC	NC	NC	NC	NC	NC	NC	C
User education	C	NC	NC	NC	NC	NC	NC	NC	C	NC
Standards Adoption	C	NC	NC	NC	NC	C	NC	NC	NC	C
High-level user requirements identification	C	C	C	NC	NC	NC	NC	NC	NC	NC
Prioritisation of product backlog	C	C	C	NC	NC	NC	NC	NC	NC	NC
Prototype development	NC	NC	NC	C	NC	C	NC	C	NC	NC
Creation of user stories	C	NC	C	NC	NC	NC	NC	NC	C	NC
Prioritisation of Sprint tasks	NC	NC	C	NC	NC	NC	NC	NC	NC	NC
Design of acceptance tests	NC	NC	C	C	C	NC	NC	NC	NC	NC
Coding prioritised tasks	NC	NC	NC	NC	NC	C	C	NC	NC	NC
Version/change control tracking	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
Code refactoring	NC	NC	NC	NC	NC	C	C	C	NC	NC
Code review with dummy	NC	NC	NC	NC	NC	C	C	NC	NC	NC
Product validation	NC	NC	NC	NC	NC	C	C	C	NC	NC
Sprint code and quality review	NC	NC	NC	NC	NC	NC	C	C	C	C
Project progress review	NC	NC	NC	NC	NC	NC	NC	NC	C	NC

Security practice / Quality practice	Security awareness training	Security analysis of user roles	Misuse case detailing	Application of security design principles	Security test design	Security coding standard adherence	Source code security reviews	Security testing	Security disclosure management	Review of security repository
Code integration & testing	NC	NC	NC	NC	NC	C	C	C	NC	NC
Next Sprint planning	NC	NC	C	NC	NC	NC	NC	NC	NC	NC
Deliverables evaluation	NC	NC	NC	NC	NC	NC	NC	C	C	C
System acceptance testing	NC	NC	NC	NC	NC	C	NC	C	C	C
Task automation	NC	C	NC	C	C	NC	C	C	NC	NC

Key: C – Compatible; NC- Not compatible

Having come up with the compatibility matrix, the next step using the adapted algorithm was to produce a Secure-SSDM that incorporates security practices adapted from the literature, without compromising the agility of the resulting method. To illustrate the use of the algorithm during the integration process, source code security reviews is the first security practice on the list with the highest agility degree of 0.83. Table 5.3 shows that it can be combined with: coding prioritised tasks; code refactoring; code review with dummy; code integration testing; and task automation. The compatible practice with the least agility degree of 0.67 is, code review with dummy. Combining these two gives an agility degree of min (0.67, 0.83) = 0.67. As this is greater than 0.5, these two can be combined resulting in the practice: Performing code and security code reviews with the help of a dummy partner. This is a practice in the Development with review phase. The rest of the security practices incorporated into the SSDM were integrated this way. Table 5.4 gives the list of practices for the Secure-SSDM. Core practices of the SSDM shown in Table 5.1, have been combined with security practices shown in Table 5.2 to give the core practices of the Secure-SSDM. The other quality practices derived from the literature remain as optional practices that are carried out to improve the quality of the developed software.

Stage I of the Secure-SSDM is elaborated here to illustrate the interpretation of methodology practices in Table 5.4. At the onset of the project, the developer starts off by educating users

on how the project will be undertaken using the methodology, at the same time, users should also be educated on issues of security. During this period, appropriate standards (both developmental and security) determined by the type of software under development, should be adopted. At this stage it is also important to carry out security analysis on user roles. The security analysis here is based on a high-level user requirements list. A set of recommended tools and techniques for executing activities in each stage of the development process is provided. The recommended tools and techniques address concerns raised by student participants during the first case study to evaluate the utility of the methodology. Since the practices recommended to build security into the software were new to most participants, they recommended the use of tools in executing these practices. Developers should refer to quality standards relevant for their software product to assess the quality of the product under development. An automated dashboard using MS Excel or any appropriate planning tool can be used to capture and keep track of user requirements. Bernabé, Navia and García-Peñalvo (2015) recommend the use of Trello or Taiga for single development environments. Trello as a tool enables the developer to organise and manage their work so that they can easily visualise their progress. In Trello (Atlassian 2019), a project is represented by a board, into which board members (project team) can be added. In a solo development project, there is usually one board member, who may work hand in hand with the user. Several boards can be created at the same time to show the various projects the developer is working on. Associated with a board, are lists and cards. A list is used to show the flow of work. which is entered in cards. A card shows the smallest unit of work in a project. A developer can create their own set of lists which they can use to visualise project progress, by moving cards across the lists. The most basic lists are those showing what tasks are in progress, pending and done (Atlassian 2019). A menu provided with the platform helps the developer to manage the named project processes. As a web-based tool, Trello provides portability so that the developer can access their dashboard from anywhere. Android and iOS apps for Trello can be used to further support portability.

Table 5.4: Secure-SSDM activities, tools and techniques

Stage	Secure-SSDM activities	Tools/Techniques/Standards
I. Management Buy-in and Standards Adoption	• Education of users on methodology & institution of security awareness programs	• Software Quality standards

Stage	Secure-SSDM activities	Tools/Techniques/Standards
	• Adoption of development and relevant security standards • Identification of users & security analysis of user roles • Establishment of high-level user requirements	• Requirements checklist (automated, e.g. Trello or Taiga/ manual dashboard)
II. Functional & Security Requirements Elicitation	• Creation of user requirements list • Creation of use cases and misuse cases • Creation of a prioritised product backlog • Creation of a WBS (up to task/subtasks) • Categorisation of subtasks • Development of prototypes	• Meeting/ interview/ document reviews • Requirements checklist • User stories • UML diagramming tools • Product backlog • Work breakdown structure • Product breakdown structure • Misuse case diagrams
III. Release and Sprint Planning	• Use of story cards to explain products • Prioritisation of Sprint tasks • Attachment of size and time estimates to tasks • Setting of the iteration duration (1 – 2 weeks) • Designing of security and acceptance tests	• Sprint backlog • UML diagramming tools • User acceptance tests (short statements showing what the system should do to be acceptable)
IV. Development with Review	• Development of code for the tasks taking care to adhere to coding and security standards • Use of version/change control tools • Refactoring of code and performing unit tests • Performing of code and security code reviews with the help of a dummy partner • Fixing identified errors • Reviewing time estimates using actual times • Product validation	• Version control system (e.g. Git, Trello) • Code refactoring • Code coverage testing tools (e.g. Jacopo) • Code review • Dummy partner (explain code to a dummy, self-dialoguing)
V. Sprint Review and Close	• Review of sprint time & code quality • Movement of finished task (s) to completed tasks • Carrying over undone tasks to next iteration • Reviewing project progress	• Version control system • White box security testing • Continuous integration • Self-dialoguing

Stage	Secure-SSDM activities	Tools/Techniques/Standards
	• Planning for next Sprint (or close project) • Performing of code integration, testing and security testing	
VI. Evaluation	• Evaluation of product deliverables & security repository update • Conducting of system acceptance test • Identification of processes/ tasks for automation (repeating tasks)	• Task/code automation tools • Security repository • Knowledge base

As Table 5.4 shows, quality practices have been organised into the stages of the primary SSDM derived through the metasynthesis performed in Chapter 2. Selected compatible security practices were then integrated with the quality practices. Only those security activities that could be performed by an individual were chosen for integration.

The foregoing integration process for designing the Secure-SSDM is summarised in Figure 5.1. First, the researcher derived the SSDM practices and security practices that could be executed by an individual from the literature. Then agility degrees for these were derived independently. After that, a compatibility matrix to ease the integration process was created. With the aid of the compatibility matrix, the security practices were integrated to the SSDM practices using a modified version of Keramati and Mirian-Hosseinabadi (2008)'s algorithm.

This section has demonstrated the rigour applied to the design of the Secure-SSDM. It has shown how quality practices and security practices were systematically drawn from the existing knowledge bases of SSDMs and secure software development methodologies respectively. It has also shown the suggested improvements on the methodological aspects of the integration process particularly with regards to the solo environment. The resulting secure solo software development methodology is this research's contribution to knowledge in the solo development environment. The next section describes the Secure-SSDM together with the tools and techniques recommended to support the developers using this methodology.

5.3 The Secure-SSDM

The version of the methodology discussed in this section is a final version. It incorporates the suggestions raised in consensus by both academic and industry participants who participated in the multiple case study to evaluate the methodology. The valuable suggestions of the three

anonymous reviewers who critiqued the submission discussed in Moyo and Mnkandla (2020) also helped to refine the methodology.

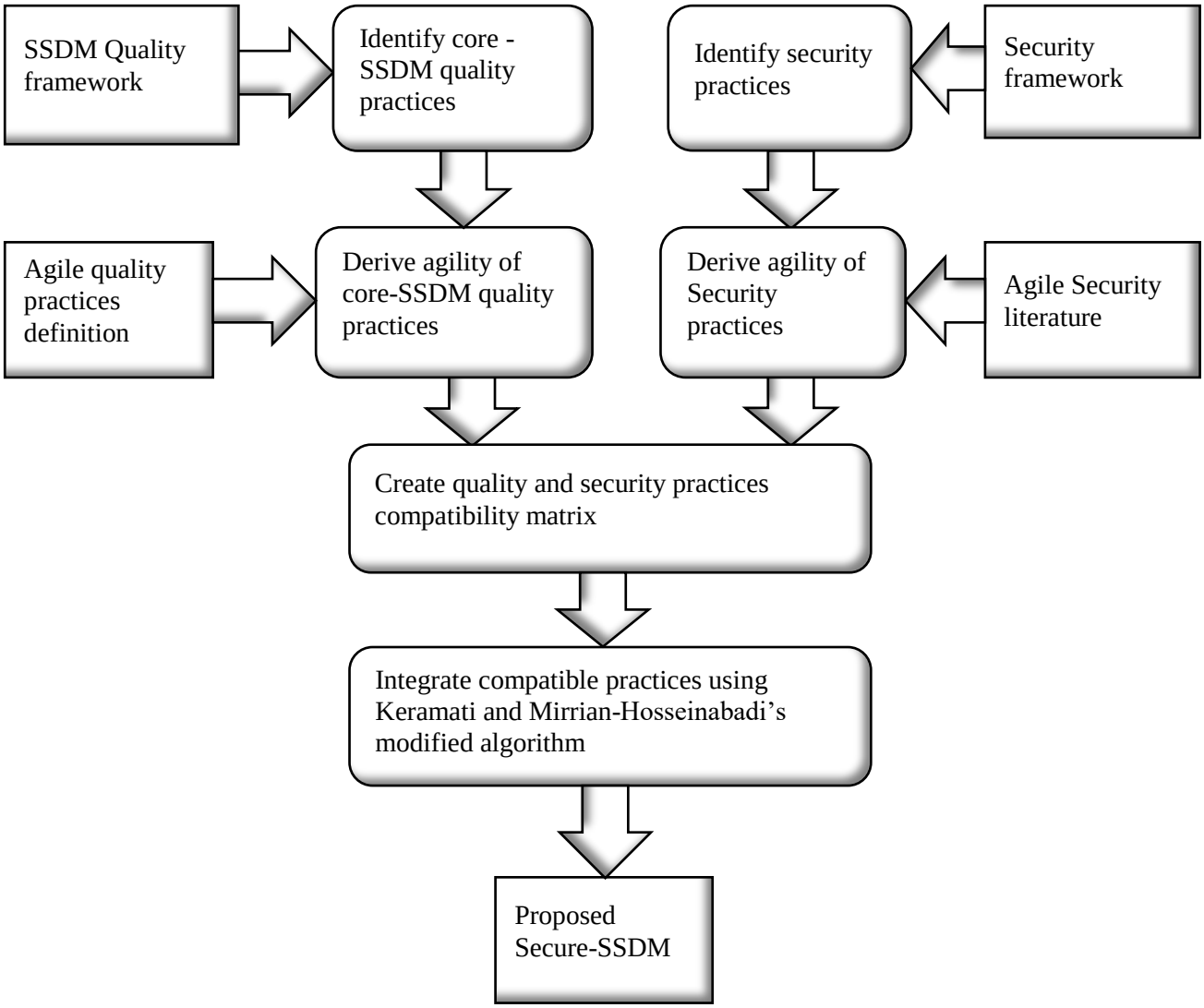


Figure 5.1: Secure-SSDM practices integration process

The Secure-SSDM emphasises on knowledge management for the benefit of the developer in future projects. While the methodology was developed iteratively, it in turn uses an iterative approach to product development. Developers using the Secure-SSDM deliver the product in increments. Whereas the methodology stages are shown in sequence, developers using the Secure-SSDM may begin a subsequent stage while working on another stage, as long as the

subsequent stages can be handled in parallel. For instance, a developer may start requirements elicitation during the process of establishing standards to be used in the project. These two processes may be interleaved. The following subsections elaborate on the activities carried out in each of the development stages of the methodology.

5.3.1 Management Buy-in and Standards Adoption

This is the stage that sets the development process in motion. The aim of the stage is to encourage user involvement in the development process. Here the developer analyses the environment in consultation with the project owner to fully understand the users' need. For projects with no particular owner the developer may discuss the idea of the project with potential users or review literature in the area or review similar software products in the market. Once the developer establishes that a need (or an opportunity) exists, their core task is to educate the users on how development will proceed, at the same time educating users on security issues pertaining to the system. Educating users on security issues is a backbone for secure software development (Rindell, Hyrynsalmi & Leppänen 2018). User training on security encourages them to think about and also suggest security requirements when it is time to collect user requirements at the subsequent stages.

Users also need to be educated at the project onset on the impact and costs associated with requirements change (Bernabé, Navia & García-Peñalvo 2015, p.688). During this stage it is also important that the developer adopts standards appropriate for the type of software under development. As a lone developer, adopting development and security standards promotes compliance with international standards as well as eases understanding of one's code in the future. These standards are shared with users as they constitute measures of both quality and security that will be used to evaluate the system at the end of the project. For non-technical clients, the developer may need to summarise the expected behaviour of the software product as defined by the standard adopted.

Assuming management and users now appreciate how development will proceed, end users' expectations from the product are identified using appropriate techniques. From the users' expectations, a high-level list of the users' requirements from the system is created. The output of this stage is an initial set of high-level requirements together with adopted quality and security standards to be used to measure both process and product success.

5.3.2 Functional and Security Requirements Elicitation

The standards adopted at the first stage and the high-level requirements collected in that stage serve as input for this stage. The main aim of this stage is to perform an in-depth understanding of system's functional and non-functional requirements from the users' perspective. Using appropriate data collection techniques, such as meetings, interviews, observation and documentation sampling, among others, the developer collects users' expectations of the system. These are captured as user stories describing the user's interactions with the system. Each user story should be accompanied by acceptance criteria (test cases) stipulated by the user (Bernabé, Navia & García-Peñalvo 2015, p.688). The INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable) acronym suggested by these authors can be used in formulating manageable user stories. Acceptance criteria for user stories serve as a guide of what is expected of the developer from the development process. During this time the developer also identifies user roles in the system, which may be captured in a checklist to visualise and simplify these. The checklist if created, is then used to define access levels on the system under development (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.27). This research recommends the use of UML (Unified Modelling Language) diagrams to model system components. These support object orientation which is the abstraction used with the Secure-SSDM. The collected user stories are therefore translated into use case diagrams. The latter are then used to perform security analysis on the users' interactions with the system, leading to the definition and modelling of misuse cases (Rindell, Hyrynsalmi & Leppänen 2017). In identifying misuse cases, users are encouraged to imagine an intruder making use of a use case (or any system component) in an illegal way. Each use case may therefore be associated with an intruder, whose intentions are captured as misuse cases. Intruders may also have their independent actions not associated with use cases, and are captured as misuse cases.

An example diagram for capturing use cases and misuse cases is shown in Figure 5.2. As shown in the figure, the user's intention is to log into the system. A possible identified threat to this activity is that of an intruder who may want to steal the user's login details to perform malicious activities against the user's data. In this case spoofing of the user's login details poses as a threat to the process of logging in. Thus, the developer needs to design security

features against this threat. Other threats to the system are identified and modelled in a similar way.

Use case and misuse case diagrams can further be detailed as suggested in Sindre and Opdahl (2005). To keep the development process lightweight, each use case description can embed its own misuse case description within its actions. The misuse case will be defined as a threat against the use case as illustrated in Table 5.5. In this case, in the use case action column, the developer captures activities that can be performed by the user for the use case under consideration. In the system services column, the developer describes the expected system response to the user's actions.

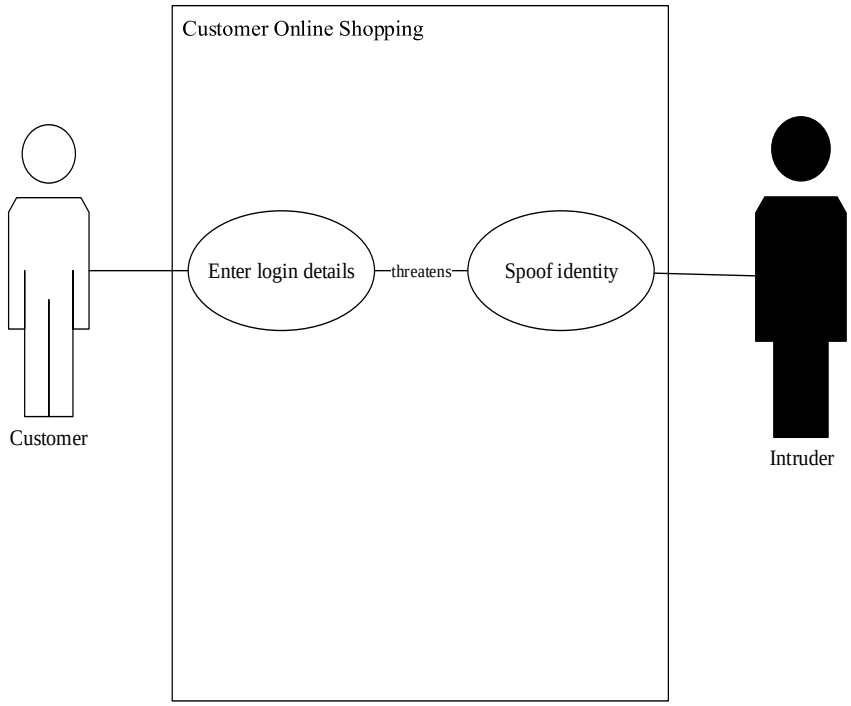


Figure 5.2: Customer login use case/misuse case

Under the intruder threat column, the developer describes security threats that are associated with each action of the user. For example, an action to prove one's identity at system log in, should be associated with a verified identity from the system, and could be threatened by identity spoofing from an intruder. Such a listing of the use case enables the developer to associate threats with user actions, so that they can build code that secures the users' actions by mitigating identified threats.

Once the user stories and use cases have been created, they are used to create a prioritised product backlog. The developer, in agreement with the user or user's representative creates the backlog showing all use cases with their associated misuse cases. In resource constrained environments, or where the development time is short, the developer needs to analyse the misuse case impacts on the system and business, so that the security risks are classified as low, medium and high. Priority should be given to high risk security misuse cases, while the low risk ones may be ignored. A backlog may be made of a number of use cases. These are prioritised to enable the delivery of high priority functionality at the beginning of the project.

Table 5.5: Embedding misuse case into use case description (adapted from Sindre and Opdahl (2005.p.37))

Use case name: Log into user profile		
User -action	**System services**	**Intruder threat**
Enter login details	Verify details	Spoof login details
…….	…….	…

A product breakdown structure (PBS) for the product under development may be created from the product backlog depending on product complexity. The PBS enables the developer to keep track of all product components and their relationships. Using the PBS, the work to deliver the components is enlisted. This can be organised in the form of a work breakdown structure (WBS). The WBS promotes product completeness (Dzhurov, Krasteva & Ilieva 2009; Pagotto *et al.* 2016), as it should be created using the hundred percent concept. The hundred percent concept means the work at level n is equivalent to work at level n-1, where n and n-1 are levels of decomposition of the WBS. However, for simple products these two models can be ignored to reduce documentation associated with the development effort. While the Secure-SSDM suggests the use of all these models, developers should choose those tools and models that promote quality at the same time enhancing their productivity, without compromising developer performance. For small software products a simple checklist may suffice to keep track of the backlog.

Using the product backlog, the developer categorises the tasks/subtasks in preparation for the definition of sprints. A sprint is a development activity that delivers meaningful functionality to the user. Developers designing complex systems can build prototypes to help them fully understand the requirements for the product and sprint. The deliverable at this stage is a prioritised product backlog with identified quality and security requirements for each

deliverable. The unique feature of the Secure-SSDM is the attachment of security requirements to user functional requirements. This entails that the developer thinks of security in advance instead of having it as an after-thought.

5.3.3 Release and Sprint planning

The prioritised product backlog from the previous stage serves as a source of items for planning at this stage. Release and Sprint Planning creates a development plan for the sprint. The task categories in the task list for the current sprint are used to create sub-tasks for the current sprint. A sprint may constitute a number of iterations that deliver internal components at the developer's site. A WBS, if it has been created, can be used to see which sub-tasks constitute what tasks. In such cases, associated with the WBS should be a product breakdown structure (PBS) showing the relationship among product components. This is true for complex projects. The PBS should be a translation of the WBS, that is, it should be clear to see how the product is produced through the WBS (Pagotto *et al.* 2016). At this point security design should be made for each deliverable associated with a task. Design should be simple enough to facilitate changes in the event that users request for such changes. The developer may use sequence diagrams or activity diagrams to understand the flow of events in each use case.

Sprint planning constitutes setting of small milestones for the project, so as to encourage development focus. Milestones mark the end of a sprint and can be used to measure project progress. For individual developers, small milestones result in frequent product delivery which in turn help to build trust with the user, at the same time promoting visualisation of development progress. As recommended by some SSDM authors, tasks in a sprint should be planned to be achievable within a duration of 1 – 2 weeks. Each task in a sprint should carry size and time estimates. This is achievable if user stories have been formulated to comply with the INVEST acronym. Developers are advised to keep track and document their performance in task execution, so that this serves as a historical database for reference in future projects. Automated tools may be used for tracking purposes to keep the process light.

During sprint planning, acceptance tests and security tests for each sprint should also be designed. These are derived from acceptance criteria formulated for the user stories. Tests are used to evaluate the quality of the deliverables at the end of the sprint. Automating these tests

reduces development effort, and serves to ensure that only tested code is integrated into the product baseline. Automated test tools such as Junit (for Java environments) or VBUnit (for Visual Basic developers) can be used for the purpose. At the end of this stage, a clear list of tasks and associated deliverables and both acceptance and security tests should be produced.

5.3.4 Development with code and security review

The input to this stage is a prioritised list of tasks. Once the tasks for a sprint are known, the developer creates code for the product component to be delivered at the end of the sprint. For enhanced productivity, developers are encouraged to use a programming language they are familiar with. Development should be carried out to comply with coding, quality and security standards adopted at the onset of the project. All code should be reviewed thoroughly, and a dummy partner can be used to play the part of a pair. Here the developer explains their code to the dummy, hoping that as they explain their code, they will identify any code that does not make sense (Bernabé, Navia, & García-Peñalvo 2015, p. 691). Besides explaining code to the dummy, the automated code and security checks provided by the programming environments suggested in Section 5.3.3 should be used to detect and deal with all coding errors. Secure coding practices such as avoidance of unsafe functions (Belk *et al.* 2011), as well as reviewing of code to identify vulnerabilities in code (Rindell, Hyrynsalmi & Leppänen 2018), should form part of the coding process. The developer should concentrate on high risk modules such as those receiving data from the outside, interfaces with other systems and access control points (OWASP 2017, p.53). This encourages the developer to deal with security issues during the development process. Just as the developer performs code reviews to identify technical debt, they should also perform source level security reviews to identify vulnerabilities in code. For critical systems developers may need to engage a consultant to review their code for both quality and security. This however while ensuring system quality may imply more financial resources are needed for the project.

All errors identified during code reviews and unit testing should be fixed before code is integrated into the baseline. At the end of the sprint, the actual and estimated times should be compared, and any differences used to adjust estimates on remaining sprint time estimates. As the product grows at the user's site it should be continuously validated at the end of each sprint, with the use of standards set during the Release and Spring Planning stage. The

deliverable from this stage is secure code with minimal, if not free of coding errors. This is ready for installation at the user's site in the next stage of Sprint Review and Close.

5.3.5 Sprint review and close

This stage marks the end of the sprint. Activities carried out at this stage transfer the developed product (or component) to the user's site. The new code is integrated to the existing code after the developer has satisfied themselves that all the quality and security standards adopted at the onset of the project have been met. The use of a version control system is highly recommended, so is the use of automation tools discussed by Driessen (2018). These include tools used to: quickly access all recently modified code files; correct the most recent commit; delete the most recent commit; and divide a commit in the event the developer detects or suspects some conflict within code components. Such tools help the developer to access the most recent work and perform corrections without taking time to browse all files. Security tests should be performed on all code before integration. All finished tasks should be moved to completed tasks, while undone tasks are moved to the next iteration. At the end of each sprint, the developer reviews project progress in consultation with the user, and adjusts plans accordingly. If the project is not yet complete, this is the time to plan for the next sprint with new information obtained from comparing the plan with the actual. For the last sprint this should mark the end of the project, therefore the review is a project review.

5.3.6 Evaluation

Evaluation marks the end of the development process. Product deliverables are evaluated against the appropriate quality and security standards adopted at project onset. A system acceptance test is conducted, pending user sign off. At this stage developers perform the following main tasks: evaluate the quality of product deliverables; conduct system acceptance test; identify processes for automation (candidates for these are repeating tasks); use the just ended project information to improve security repository. Apart from enhancing the developer's security skills, the repository helps to show which parts of the system need maximum security.

The Secure-SSDM flow is shown in Figure 5.3. The key tasks performed in the various stages are briefly summarised in the diagram. Developers identify key users who should include project sponsors and educate them on the main processes of the methodology and on the importance of participation during the development process. Thereafter the developers

working with the users identify both the functional and security requirements of the product, and development proceeds as explained in the sections 5.3.1 to 5.3.6.

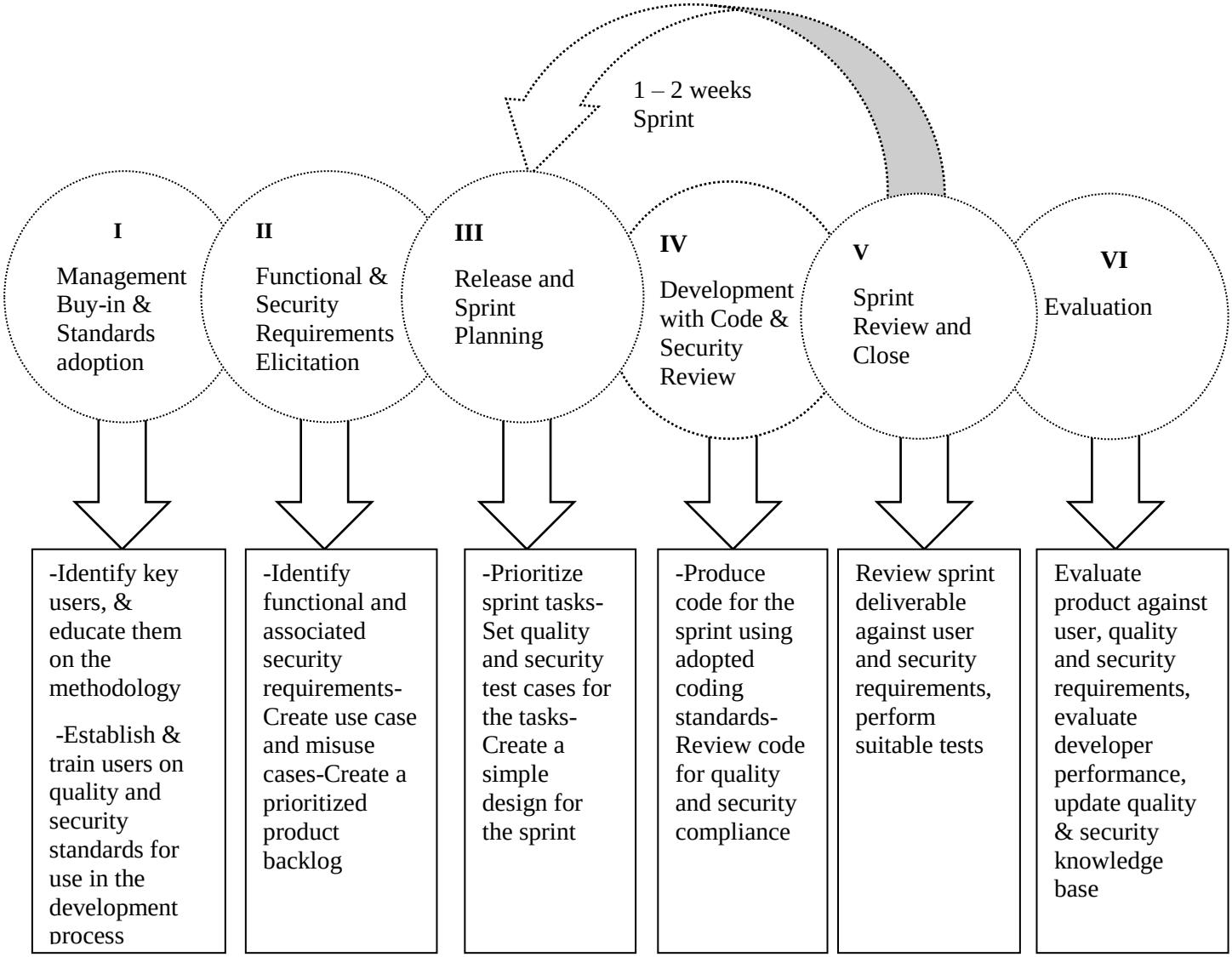


Figure 5.3: Secure-SSDM stages summary

5.3.7 Modelling the Secure-SSDM

On testing the methodology with second year Computer Science students at the National University of Science and Technology (NUST), Zimbabwe, it was evident that there is a need to provide a comprehensive model of the Secure-SSDM, with appropriate tools and techniques to support each stage, particularly for security practices. Most students did not have prior knowledge of these, neither did they have knowledge of the appropriate automation tools to use with the methodology. In this case the student participants represent novice developers. Such developers would need an appropriate tool and model support to enable them to undertake the practices recommended in this methodology. Besides documenting the tools and techniques to be used with the methodology, it is important to specify the deliverables expected on execution of the various activities. The EPF Composer served as an ideal tool to document the methodology, as it supports the documentation of roles, processes, and tools for use by the various roles. It enables method engineers to package knowledge required for a particular process so that developers can use the tool as a knowledge base (Eclipse Foundation 2018).

Modelling the Secure-SSDM with the EPF Composer facilitates usability and updatability of the artefact, as the developer can easily update the activities defined within the methodology after project execution, so that they document activities that work within that project environment. EPF Composer therefor acts as a knowledge management tool in this solo development environment. Various versions of method components, method-plug-ins and tasks can be created and managed for the various development projects the developer works on. Two screenshots from the Secure-SSDM are shown in Figures 5.4 and 5.5. Figure 5.4 shows the screenshot of the main page of the Secure-SSDM method library with the various method plug-ins for the library. In this case the various stages of the Secure-SSDM were created as method plug-ins. Each of the stages had its content defined describing the work products of the stage, the necessary skills required and appropriate guidance showing how specific development goals are achieved.

Figure 5.5 shows the first two stages of the Secure-SSDM defined as method plug-ins of the main method library. The two stages are Management Buy-in and Standards Adoption, and Functional & Security Requirements Elicitation. In the diagram the prioritised product backlog work product is highlighted, displaying the description of the work product on the artefact description display window on the right. The work products of the Functional &

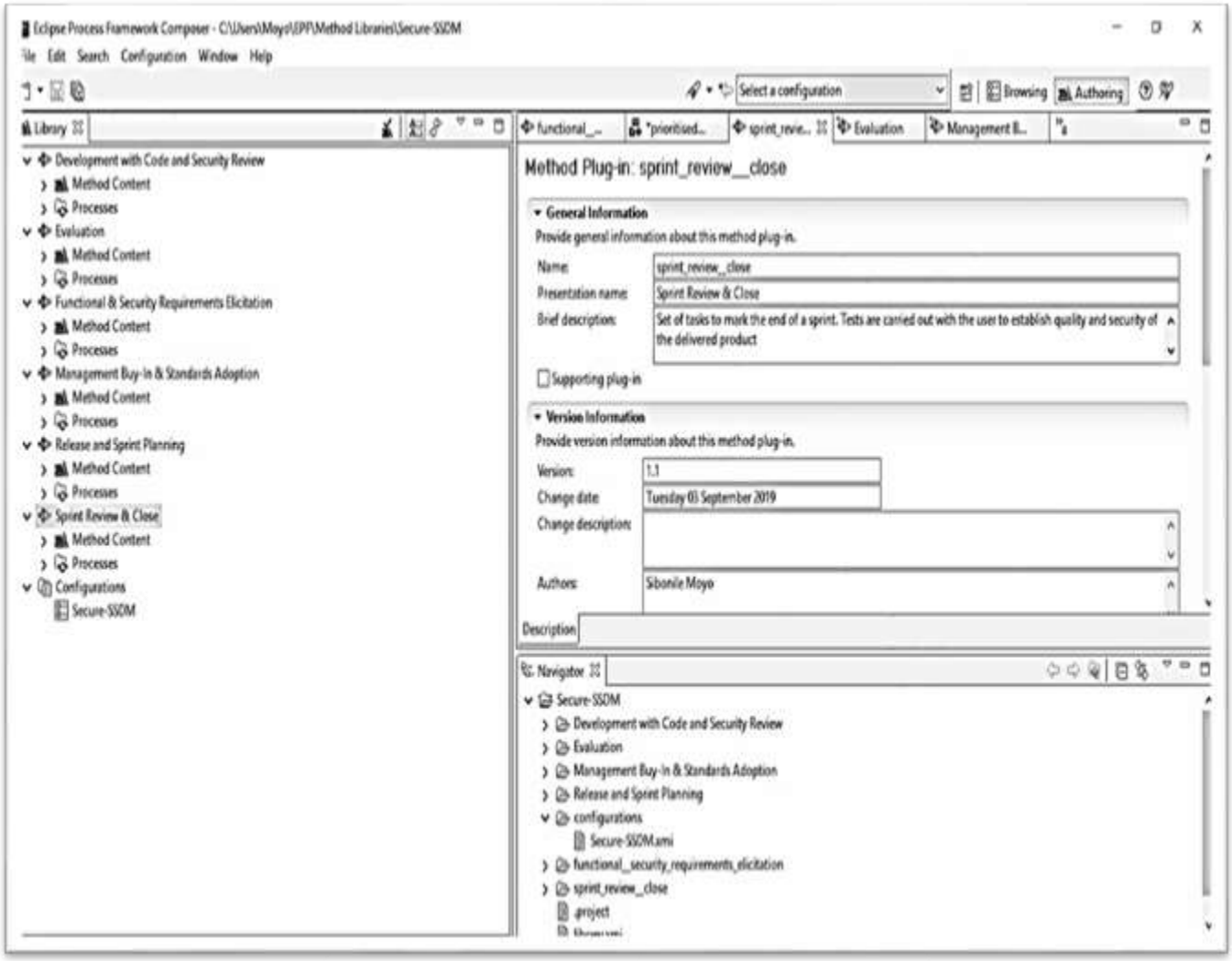


Figure 5.4: Secure-SSDM stages definition in EPF Composer

Security Requirements Elicitation stage are shown as the prioritised product backlog, use case/misuse case diagram and use case description. Two roles are defined for this stage, the developer as an analyst, and the user as an information source. Defining the roles separately enables the developer to differentiate their role as an analyst and their role as a developer.

The EPF Composer provides a flexible way of defining and publishing methodologies. While for a solo developer, publishing the methodology is not an essential aspect as the developer works alone, the tool makes it easier for the developer to communicate with the user, reminding them of their obligation, the delivery dates of the product components and the standards agreed upon to accept the product. Its flexibility also enables the developer to adapt the process to suit the kind of software under development. It also helps as a knowledge management tool in keeping the various versions of the method plug-ins.

The developer can revise the knowledge base as they discover new ways of executing the practices. The newly discovered or improved practices may be created as a revised version of

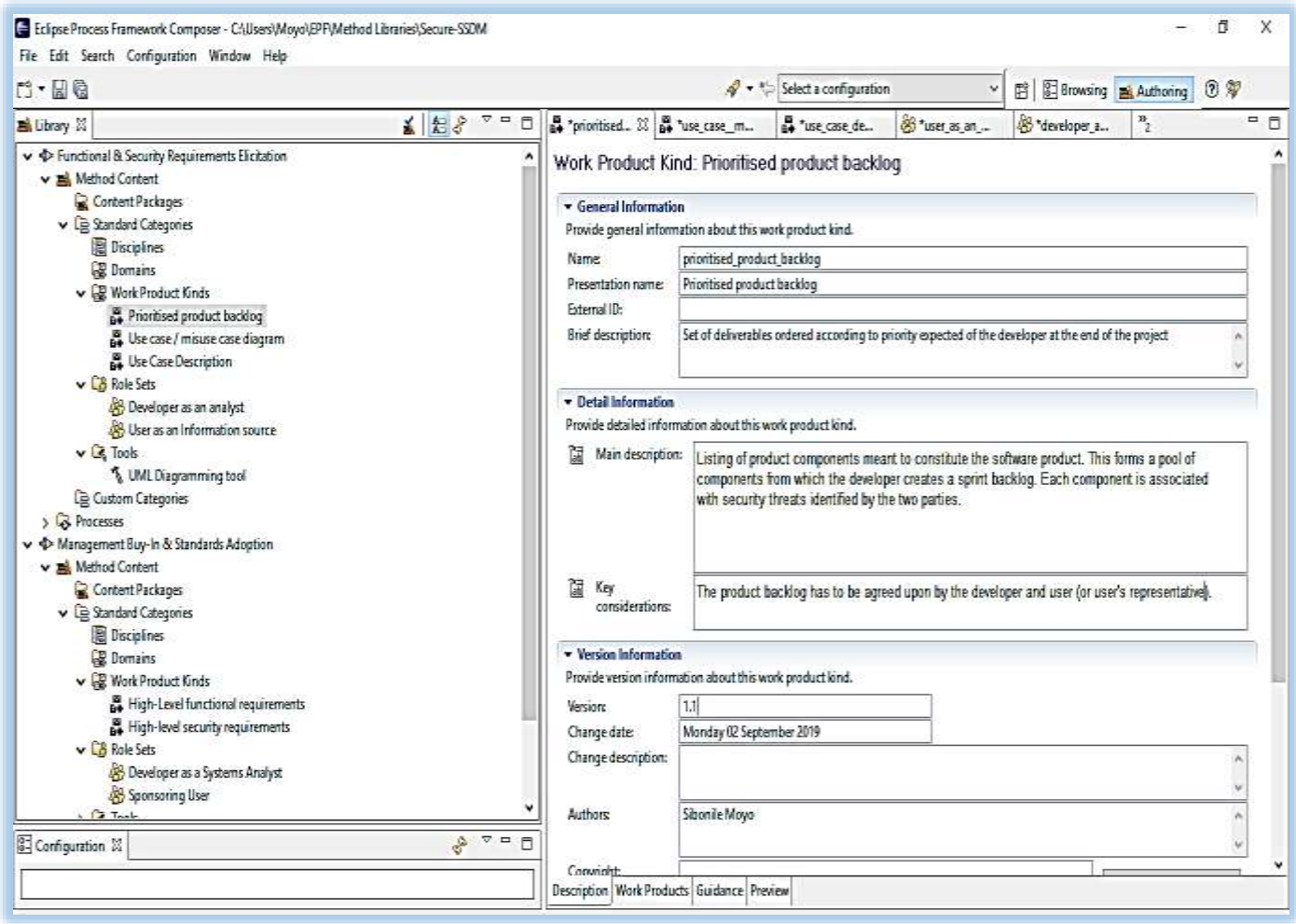


Figure 5.5: Defining prioritised product backlog in Secure-SSDM

the current. Figure 5.5 shows version 1.1 of the description of the prioritised backlog together with the date of creation.

5.4 Secure-SSDM demonstration

The key success factor of DSR is the demonstration of the utility of the artefact through using the artefact to solve a real problem in the area in which it is designed to work (Peffers et al. 2008, p. 55). Since the objective in this research is to build a methodology that enables the development of software products that meet users' functional and security requirements, there

is need to demonstrate this claim. This should be through using the Secure-SSDM to develop software products, and testing whether the developed software products are secure and meet the elicited user requirements (Walls, Widmeyer & Sawy 1992). Besides testing the resulting software products, demonstration should also prove the applicability of the methodology practices in developing software. An appropriate demonstration of this artefact's claim is to use the methodology at an individual level to develop the software. To that effect, the utility of the Secure-SSDM was demonstrated through a multiple case study. In the first case study, thirty-nine undergraduate students participated in using the methodology to develop individual mini-projects during the semester of January 2019 to May 2019. The students who were in their second year of a four-year Honours Degree in Computer Science were assigned areas from which to develop software systems of their choices to solve real-world problems in the community. This was part of the course requirement in a course, Computing in Society (course code SCS 2206) that they do during this year of study. The students were tasked to identify real projects and customers in the areas of Education, Health, Business, Government and the Environment, that they could work with to establish needs or problems that could be solved through the development of software.

This group of students was found favourable for the case as they had already done two programming courses, one in their first year and the other in their first half of the second year. They had also done a course in software development methodologies, equipping them with the skill of using software development methods in building computer software. The students in this case therefore were taken to represent novice software developers.

Three industry developers participated in the second study. The developers had a minimum qualification of a degree in Computing (Computer Science and Information Technology), with an average software development experience of four years. The second study was also designed as a summative evaluation to check the capability of the quality and security practices embedded in the Secure-SSDM to produce both high quality and secure applications. Two developers were working on new software products, while the third developer used the methodology to perform an upgrade on an existing product, they had previously developed. The methodology was explained to the developers at the onset of the study, and frequent consultations were made on their progress. The details of the two case studies are discussed in Chapter 6.

5.5 Chapter Summary

This chapter has detailed the design of the Secure-SSDM, giving explanation of the practices in each phase of the methodology. Care was made to produce a befitting design that embeds quality and security promoting practices within the methodology. The extant literature provided a rich source of both quality and security practices. An adapted version of Keramati and Mirian-Hosseinabadi (2008)'s algorithm provided a systematic means of integrating security promoting practices with the solo software development practices, taking care to retain the agility of the resultant methodology.

The use of the methodology by the undergraduate students provided a means for formative evaluation, and the results of the formative evaluation provided input into improving on the methodology design. A list of tools and techniques were discussed to address the knowledge gap of the students, who represent novice developers. A further refinement was made based on the feedback obtained from industry participants.

The following chapter discusses the demonstration and evaluation of the Secure-SSDM. The demonstration section details the two case studies carried out to demonstrate the utility of the methodology in designing and implementing quality software products. The evaluation presents the results obtained from the multiple-case study and the theoretical evaluation.

CHAPTER 6 SECURE-SSDM DEMONSTRATION

6.1 Introduction

In Chapter 5 a blue print of the proposed Secure-SSDM was developed, followed by a detail of the stages in the methodology and a representation of the artefact using the EPF composer. Section 5.2 of that chapter elaborated on how quality practices from the existing SSDM knowledge base were identified and systematically integrated with security practices drawn from the existing secure software development methods. The Secure-SSDM was then detailed in Section 5.3, where the methodology stages with associated tools and techniques in each stage were described.

This chapter discusses the demonstration and evaluation processes performed to establish the utility of the proposed methodology. As proposed in Chapter 4, the Secure-SSDM is designed to be lightweight to encourage its uptake by independent developers. Significantly, it is designed to enable quality and security in the developed software products. Evaluation therefore seeks to demonstrate the utility of the methodology to that effect. The goal is to establish the usability and effectiveness of the practices embedded in the Secure-SSDM in designing and implementing quality and secure software products.

In demonstrating and evaluating the utility of the Secure-SSDM, a DSR perspective to evaluation was adopted. According to the DSRM adopted in this thesis, the evaluation process is usually conducted in parallel with the demonstration process. Evaluation may take any of the following forms: comparing the artefact's functionality with its originally set objectives, carrying out a satisfaction survey from the target audience or use of logical proofs among others (Peffers *et al.* 2008, p.56). In this thesis the last two forms of evaluation are conducted to promote rigour in the evaluation process.

Characteristically, evaluation is an iterative process which starts at the design stage of the artefact. As the researcher contemplates on what components to bring together to create the artefact, mental evaluations of the components take place (Vaishnavi et al. 2017, p.29). The Secure-SSDM was created incrementally and iteratively, with rigorous mental evaluations performed on each increment. The first rigorous evaluation was undertaken in Section 2.5, of Chapter 2. In that section a meta-synthesis was conducted to systematically integrate various quality practices drawn from existing SSDMs. The quality practices formed the building blocks of the primary Secure-SSDM. Meta-ethnography (Noblit & Hare 1998) was used in

the synthesis to interpret, and translate the study practices into each other, so as to obtain a consensus view of the quality practices drawn from the methodologies.

The second rigorous mental evaluation of the Secure-SSDM is detailed in Section 5.2. In that section, quality and security practices were evaluated for their agility using an algorithm formulated for the purpose. Only those practices that had their resulting agility degrees higher than 0.5 were incorporated into the Secure-SSDM. The 0.5 threshold used in that case is recommended by Qumer and Henderson-Sellers (2008) as an acceptable minimum agility value of any practice or process considered as agile, based on a scale of 0 to 1. One (1) in this case is the maximum and zero (0) is the minimum degree of agility. A practice with a value of 0 to less than 0.5 is heavyweight and that of 0.5 to 1 is lightweight. These mental evaluations thus form the formative evaluation that is characteristic of DSR.

Summative evaluation of the Secure-SSDM was performed both empirically and theoretically. A multiple case study conducted with both student and expert solo software developers was used for empirical evaluation, while the 4-DAT model was used for theoretical evaluation. Student developers were drawn from a university setting, while expert developers were practicing industry developers. In the following sections and sub-sections, the demonstration and evaluation processes of the Secure-SSDM are detailed. Section 6.2 demonstrates the use of the Secure-SSDM in a software development project. Section 6.3 explains the academic case study and the results obtained from the study. Section 6.4 details the industry case study and subsequent results. Section 6.5 presents a cross-case analysis of the multiple case study results. Section 6.6 discusses threat for validity and how these were addressed. Section 6.7 presents the theoretical evaluation of the Secure-SSDM. Section 6.8 deliberates on the results of the evaluation and recommends improvements for the future. Section 6.8 concludes the chapter.

6.2 Demonstrating the utility of the Secure-SSDM

Demonstration proves that the artefact works for its intended purpose. It entails using the artefact to solve a real-world problem in the area of its application. The Secure-SSDM was applied in varied conditions both in industry and academia to solve real world problems. This section details a project undertaken by an industry developer to design and implement a web-based application that facilitates the posting of announcements in an educational institution. This application was chosen for demonstration due to its accessibility to the researcher.

Demonstrating the use of the artefact by a representative of the intended audience serves to prove from the user's perspective that the artefact works.

In this project a lone developer used the Secure-SSDM to design and implement a software product aimed at replacing the university email system for internal messages that require immediate response and tracking. Apart from using phone calls, employees send email notices to each other through the conventional emailing system hosted by Google, for both internal and external communication. All emails go through the email server and have to comply with both the organisational and Google standards and policies. This means the emails are subject to Google policies which include granting Google the rights to scan the emails. The drawback of this approach is that the notices are subject to unnecessary scrutiny at the two levels, internal and external. Further, there may be delays in communication if the email server is down. In some cases, some urgent messages may go unnoticed or may be ignored in busy days.

The developer sought to solve the delays and bottle necks associated with the conventional e-mail approach by developing a web-based system that facilitates the sending of short messages between employees. The system allows each employee to log in and check for any messages intended for them for the purposes of responding and acting on the message. This application is designed to facilitate communication and collaboration between various employees. Employees can check on each other's progress if they are jointly working on a particular task. An employee can easily check if a certain task has been attended to, and if not generate a reminder to the recipient. This system works more or less like an electronic task ticketing system, but is mainly a communication platform as opposed to the task tracking focus of ticketing systems. The version of the system detailed in this thesis has been kept simple to demonstrate the core practices in the Secure-SSDM. The system is termed the Internal Communication System (ICS).

6.2.1 Conceptualising the ICS

A developer using the Secure-SSDM starts by familiarising themselves with the organisation for which the software product is developed. In this case since the developer was part of the employees, familiarisation was an inherent process. The ICS was the developer's idea to improve communication among university employees. Two departments of the university were chosen for piloting the system. Stakeholders from the selected two departments were invited to a short meeting. The meeting was intended to share the idea that the developer had.

Stakeholders in the meeting included a representative of the head of the university's information and communications technology services (ICTS) department and two representatives, each from one of the departments selected for the purposes of piloting the project. The researcher took part in this meeting as an academic stakeholder. This facilitated observation of the development process. After sharing the idea, the developer gave a summary of the Secure-SSDM and how the stakeholders would be involved in the development process. Stakeholders gave their suggestions on what they would require from such a communication system.

In this case arranging for the meeting was easy for the developer since there was already a working relationship between the two departments involved in the pilot project. As per the developer's advice, it was agreed that the Web-services standard and university communication policy be adopted as the standards to be adhered to during the development process. These would contribute towards the non-functional requirements of the ICS. During the meeting the developer documented all the agreed upon requirements using a word processor.

6.2.2 ICS functional and security requirements elicitation

The developer used the same meeting to capture the requirements of the system. As suggested in the stages of the Secure-SSDM, the stages can be done in parallel depending on the environment and type of system under development. In this project, Management buy-in and Standards adoption (Stage I) was done in parallel with Functional and security requirements elicitation (Stage II). To gather requirements, each meeting participant was asked to write down their expectations from the system in the form of a story. Associated with each story participants were also asked to imagine what an intruder would do to disturb the smooth flow of the user's actions. This was then put down as the however part of the story. For example, a participant wishing to post a message to another employee produced the story:

"As a user I would like to post a message to a colleague, however an intruder may distort or delete my message".

The developer extracted all the stories into a template prepared for the purpose. The team agreed that the requirements captured in the meeting were key to the functionality of such a system. The captured requirements are shown in Table 6.1.

Table 6.1: Template for capturing user requirements

User	Expectations for communication	Identified threats to smooth communication
User A & B	Post messages	Illegal posts, failure to access system, loss of messages, wrong posts
	Update/delete messages	False update, lost update, illegal delete
	Respond to messages	False response, delete response
	View messages	Illegal view of messages, failure to view messages
Admin	Create/register user	Unauthorised user creation, illegal access/ stealing of user credentials
	Delete message	Wrong message deleted, illegal deletion of post

Using Table 6.1, the developer created a use case/ misuse case diagram representing the overall system requirements. Misuse cases were modelled using the threat column of the table. Only threats from an outsider were modelled to avoid mixing user errors with security threats. The composite use case and misuse case diagram was created on a sheet of paper during the meeting. The agreed upon use case diagram was later refined using MS Visio, and adopted as a set of requirements for the ICS. Figure 6.1 shows the use case diagram of the ICS.

As shown in Figure 6.1, three types of users were identified in this system. The user is any authentic employee that may need to communicate with another employee of the organisation. Employees can send messages, view messages, respond to messages and delete messages. A message can be deleted if it was generated in error, or has been resolved. A message can only be deleted by the originator, recipient or the administrator. To access the system, users have to be registered on the system. This is so, so as to restrict any employee posting messages on this platform. Modelling the system using use case diagrams helps the developer to identify those use cases that may be used in other use cases. The <<extend>> and <<include>> associations as described in (Ambler 2001, pp.190-193) facilitate this. The update use case in this case has been modelled as an extend use case of the post message use case.

The second type of user in this system is the administrator. The administrator in this system is a representative member of the ICTS department responsible for manning the ICS. The administrator can add users into the system, view posts from any user, and delete posts from

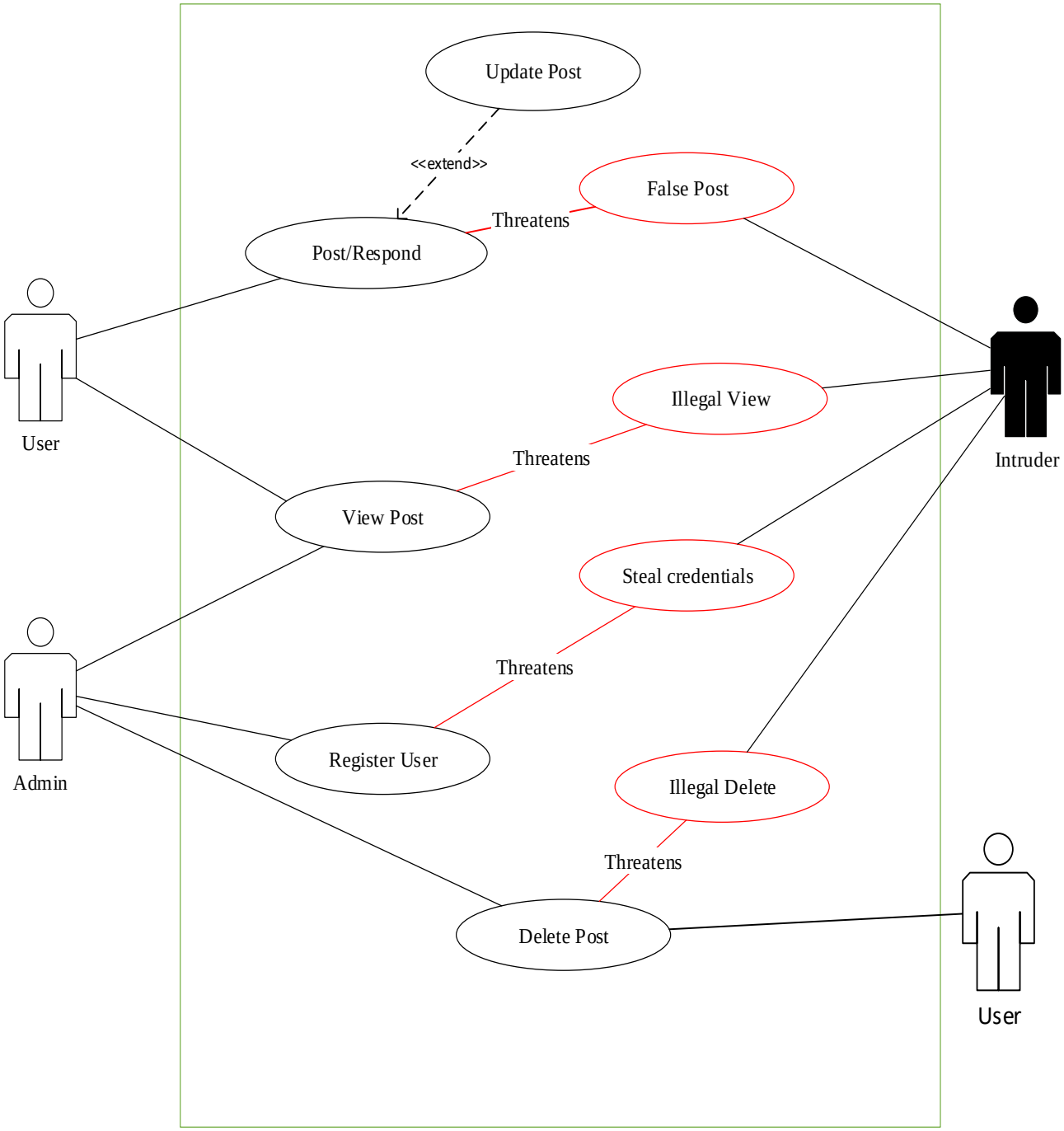


Figure 6.1: ICS use case diagram

the system. Any posts remaining in the system for a certain period should be archived by the administrator. This is in compliance with the university communication policy. When the administrator adds a user, they give them rights according to their role in the institution.

In modelling the security aspect of the ICS, a third actor in this system was identified as an intruder. An intruder is any person that may want to access the system illegally. Some information sent between employees can be highly confidential and needs to be protected from both insiders and outsiders. As shown in the diagram, an intruder may render the system insecure if they view messages not intended for them. A false post may also send false messages to employees therefore the system needs to be protected from such.

To elaborate the use case /misuse case diagram a tabular listing of these as described in (Sindre & Opdahl 2005) could have been produced. However, in this case since the system was small and straight forward, the developer decided to minimise documentation. After the stakeholders had agreed on the functional and non-functional requirements for this version of this system. It was agreed that the developer starts by designing the database for the system. Since the system involves sending of data, all messages sent between employees should be captured into a database. After the database design, the administrator module was to follow and the user modules to conclude. These served as a product backlog for this system.

6.2.3 Release and sprint planning

During this phase the developer creates a plan for executing the tasks selected for the current sprint. The order of task priority in this case was not changed from the one agreed upon during the stage, Release and sprint planning. The first task entailed coming up with a database to store user credentials, and messages posted through the system. In this case, the database was identified to have two entities, the employee and the message. There was no need to create a work-breakdown structure for this system as the developer perceived it to be a simple system with minimal tasks. The database was implemented using MySQL. This is the platform the developer normally uses for most of their projects, and is a free and open source platform.

Using the product backlog, the three sprints were planned to last a week each. However, in some cases where the developer was doing normal work after working hours, the sprints lasted longer than planned. In planning for the sprints which were dedicated to the administrator and users' modules respectively, the developer designed test cases for each of the use cases. In

this case a word processor was used to tabulate the tests. Table 6.2 shows test cases developed for the use cases identified in Figure 6.1.

Table 6.2: Test cases for each ICS component

No.	Test	Action	White box test result
1	Login	Use correct credentials	Login success
	Login	Use wrong credentials	Login fail
2	View Messages	User with messages	Messages displayed
	View Messages	User with no messages/ intruder	No messages displayed
3	Respond to message	When no response created	Allows response to be created
	Respond to message	When response is there	Displays response
4	Create message	If authentic user	Create Message
	Create message	If not authentic user	Deny access
5	Create user	If admin	Create user
	Create user	If not admin	Deny access
6	Archive user	If admin	Archive user
	Archive user	If not admin	Deny request
7	Archive message	If message author/ recipient/admin	Archive message
	Archive message	If not message author/recipient/ admin	Archive request denied
8	Update message	If message author	Update message
	Update message	If not message author	Update denied
9	View system log	If not admin	Viewing denied
	View system log	If admin	View log

Planning also involves coming up with simple designs of the system. As recommended in the Secure-SSDM, the developer should always opt for the simplest design. Before implementing each use case, the developer created sequence diagrams to depict logic of each process carried out by user on the system. A sample of sequence diagrams for the use case modelled in Figure 6.1 are given in Figure 6.2 to 6.5. The demonstration in this section concentrates on the use cases that are core for the functionality of the system. Figure 6.2 illustrates the sequence diagram showing the sequence of events expected from an attempt by an authentic user with no posting rights to post of a message in the system. Such a user while granted access into the system, should be denied the right to post a message. To keep track of failed and successful post attempts the developer kept a log of these. These were stored in the databases. This should assist the administrator with the system statistics. The number of failed attempts should be an indicator of how secure the system is, assuming the post module is working as expected.

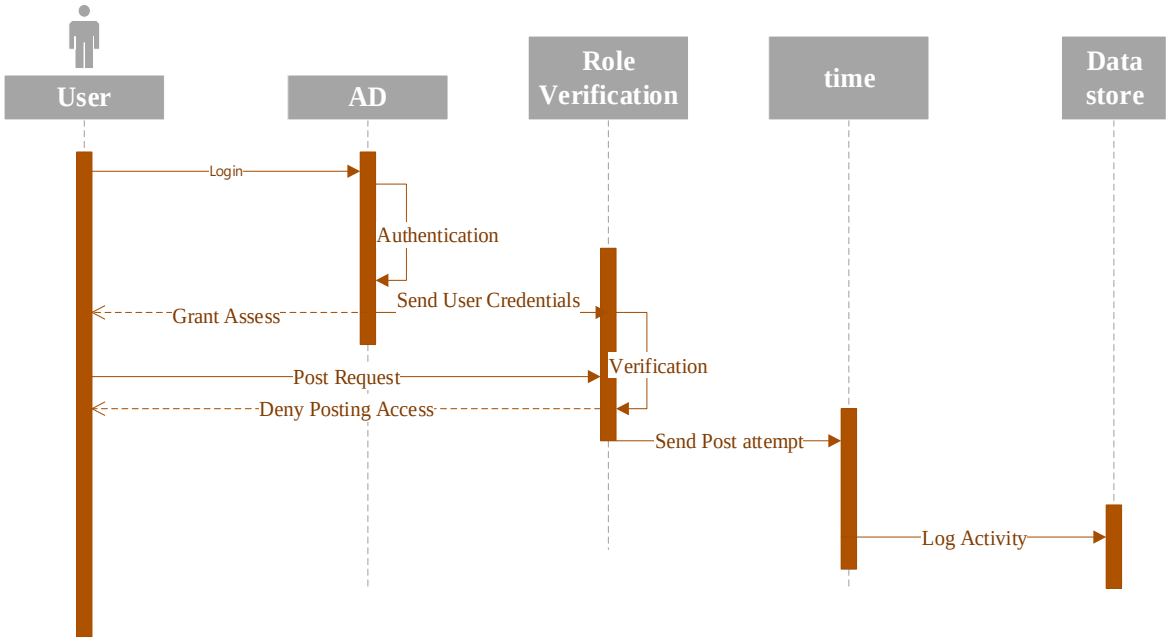


Figure 6.2: Denied post request sequence diagram for an authentic user (courtesy of Participant A)

Modelling the system with sequence diagrams enables the developer to verify the logic of processes involved in executing a given use case. A sequence diagram may model a part or a complete scenario of a use case. Sequence diagrams serve to link the analysis and the design stage of a system. In Figure 6.2 the sequence diagram visualises the processes what should take place during an attempt to post an announcement or send a message by a user with no posting rights in the system. As shown in the diagram, after a successful login, before any user can post an announcement, a request to post is generated. Their credentials are used to check their rights in the system through the role verification module. Since the user has no rights, posting is denied and the attempt to post is logged in the system. This way, only authorised users can post announcements or send a message to any user. The developer has enforced security, through the login process and the role verification process. This two-level authentication scheme deals with the case of an intruder who manages break into the system with the aim of making a false post.

Figure 6.3 depicts the set of processes involved when a user with rights requests to post a message through the ICS. The same set of processes are followed, but in this case since the user has posting rights, these are granted and posting is successful. Similarly, a log is generated for security purposes.

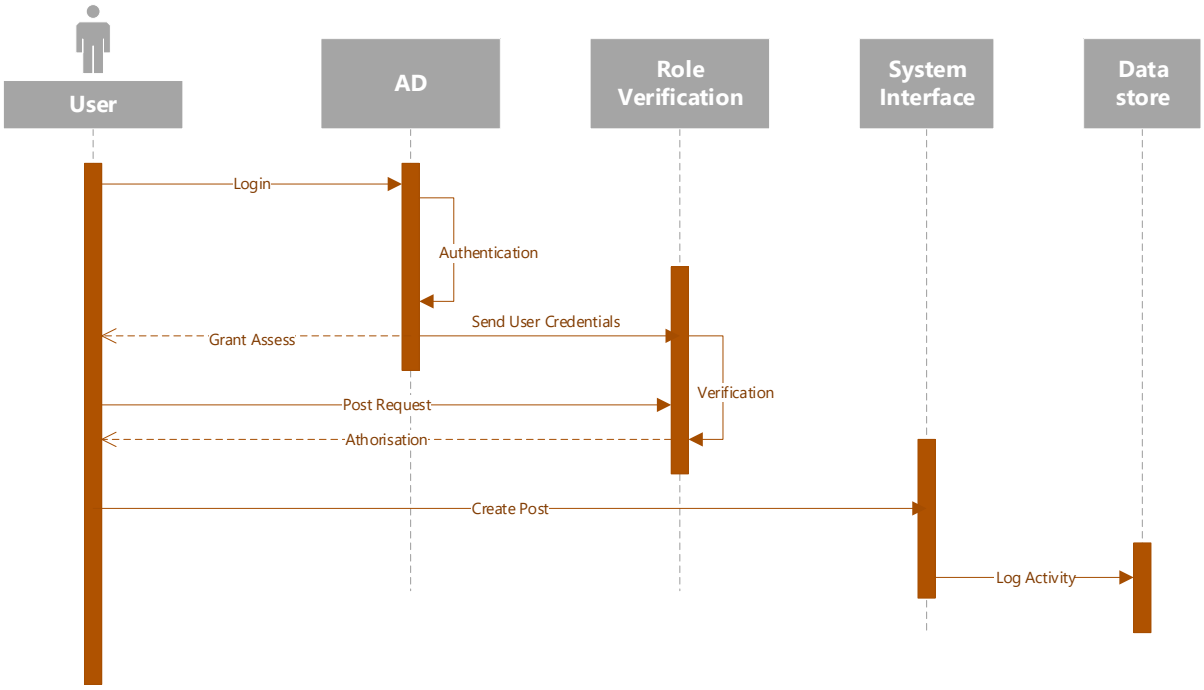


Figure 6.3: Message posting by user with rights (courtesy of Participant A)

The other key feature in the ICS is that only users registered in the system can send message posts. User registration in this case is performed by the administrator. On creating users, the administrator grants them rights according to the university policy. Figure 6.4 depicts the process of enrolling a user into the ICS.

Only the administrator has the right to create users. To ensure security for registered users, each user's credentials are encrypted. Each user has roles and rights associated with them. These are stored in the database, and are used to grant users the various kinds of access. A created user receives their credentials through their conventional email. These constitute the user name and password. On reception of their credentials, users are advised to change their passwords to enhance system security. Whereas the system facilitates the registration of users by the administrator, it should deny non administrators the right to register users. Figure 6.5 shows the sequence of events for a user denied access to register a user. The sequence diagram shows that no other user can add a user without the necessary credentials.

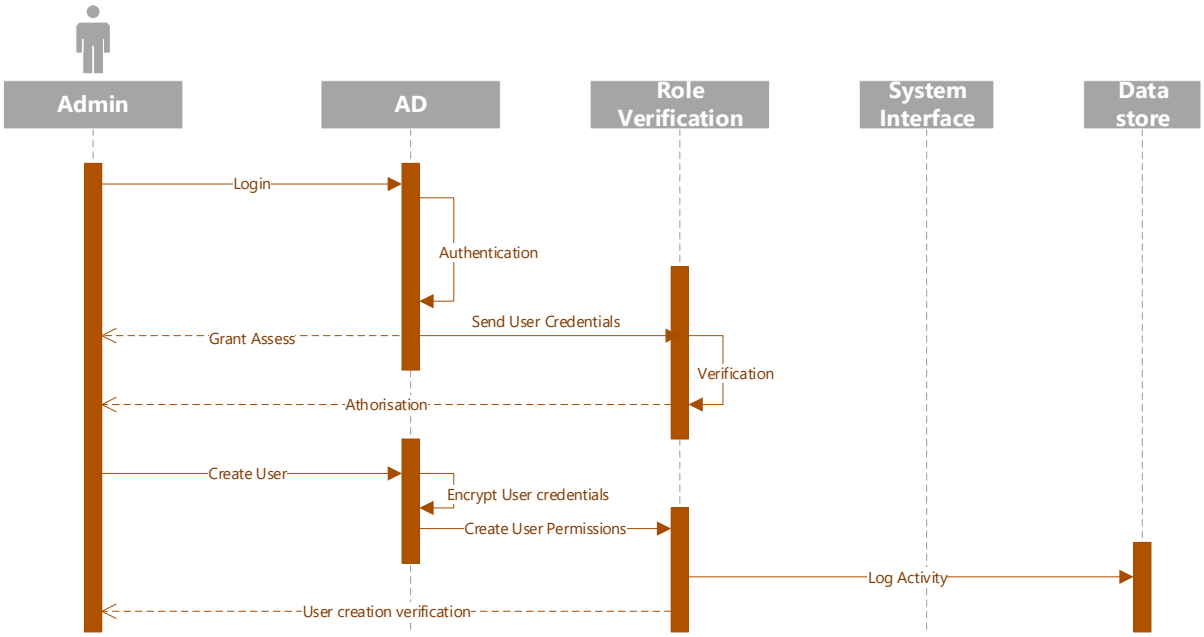


Figure 6.4: Creating a user by an administrator with rights (courtesy of Participant A)

Other sequence diagrams for the rest of the system were developed in a similar manner. In this case the developer produced two sequence diagrams for each use case depicted in Figure 6.1. One use case was drawn to depict normal flow while the other was produced to depict the intruder or unauthorised user scenario. One of the objectives in the design of the Secure-SSDM was to keep the artefact as lightweight as possible. Using sequence diagrams to model the system means the developer can use the same diagrams for both for analysis and design(Ambler 2001, p.208). After the developer had produced the sequence diagrams for the captured use cases, the next activity was to develop code to implement the processes depicted in the diagrams.

While the developer only used sequence diagrams to depict the design of their system, other forms of design diagrams like the activity diagrams would serve to clarify the user's expectation from the implementation in a similar manner. Activity diagrams can also be used to model the flow of events in the system. These are important in cases where conditions in the environment determine the next sequence of events. An alternative way that the developer could have used to model the logic of posting a message is given in Figure 6.6. In this example

a user can only post a message after they have been authorised to do so. Otherwise the system terminates without the user posting the message.

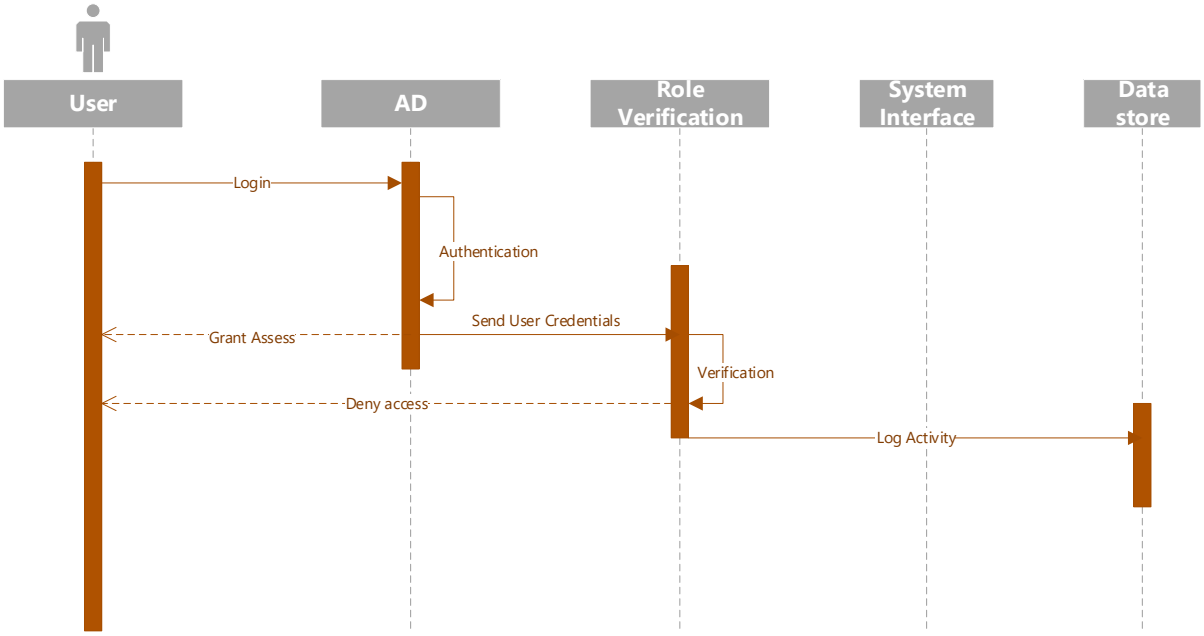


Figure 6.5: Unauthorised user attempt to register a user (courtesy of Participant A)

Figure 6.6 shows that a user wishing to post a message logs onto the system, using their credentials. If wrong details are entered, the user is asked to enter correct details. The activity diagram in Figure 6.6 shows the events involved in posting a message. This could be posting of a message by an administrator or any user. Activity diagrams are important in that they can be used to depict sets of processes covering a number of use cases in the system. The activity diagram demonstrated in this case shows a number of scenarios. In the first scenario a user can log onto the system with correct details and request to post a message. Permission to post is granted based on their credentials. A user with no rights is denied access and the system stops. A user submitting wrong credentials at login is given an opportunity to re-enter these, and if correct the system proceeds as explained before. A user supplying wrong results forces the system to stop.

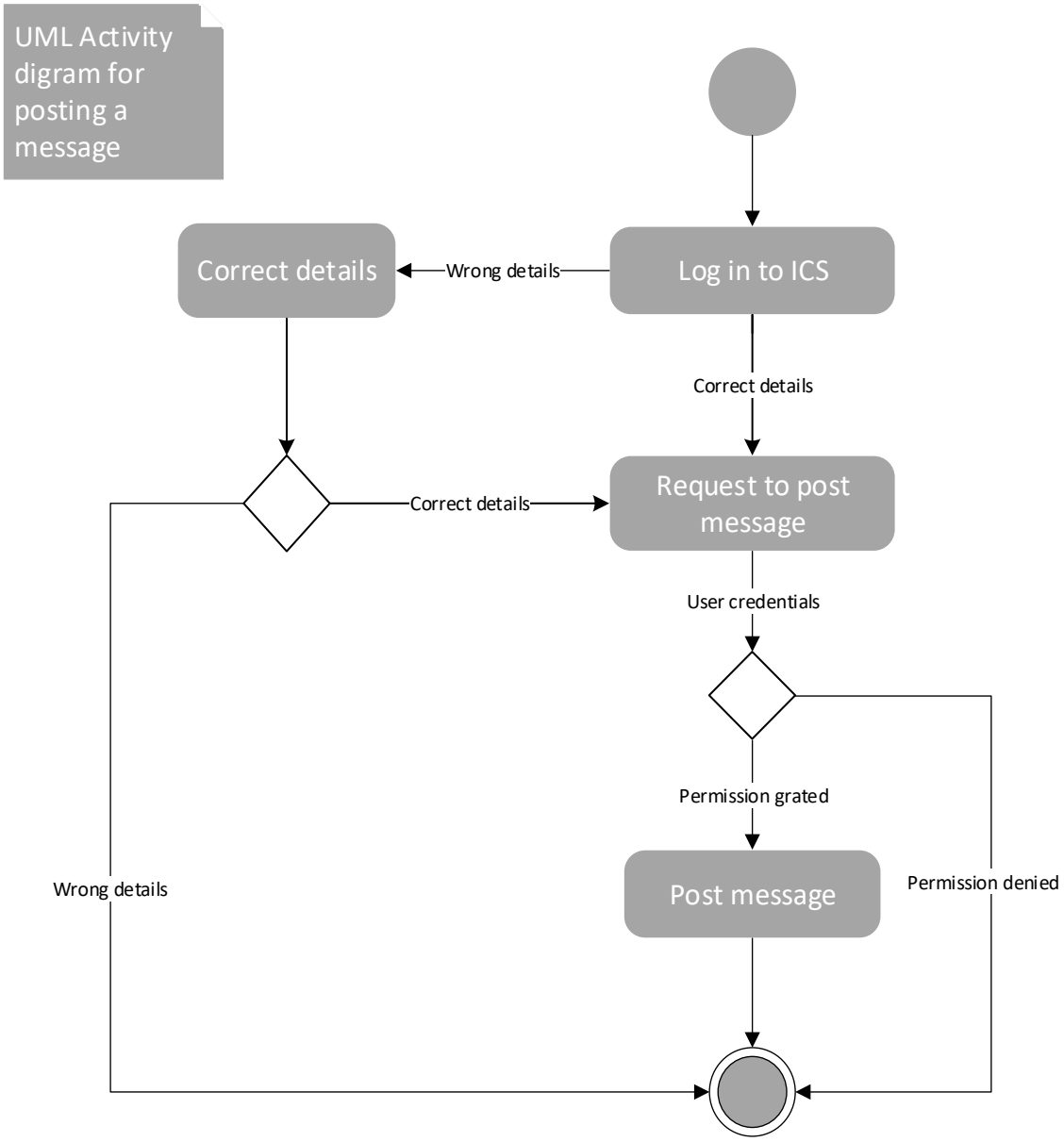


Figure 6.6: Activity diagram for posting a message

6.2.4 Development with Code review

At this stage the developer is set to translate their design into code. As recommended in the Secure-SSDM, developers are advised to use a development environment they are familiar with. In this project, the developer used Visual studio 2017 as the development environment. As an integrated development environment Visual studio offers a number of benefits to a lone

developer. Among the options available are those to set tests for your code, run the test and analyse both your tests and code. The test cases defined during the stage of user stories were set at the onset of the implementation of each user story. The code was written in C# and the database used was MySQL.

6.2.5 Sprint close and review

This fifth stage marks the end of a sprint. Three sprints were set for the ICS project. The first sprint was set to deliver the database which was designed and implemented using MySQL. To enable the demonstration of the structure and functionality of the database, all user representatives had to install MySQL on their machines. The database was up and running at the end of the first sprint. The second sprint was dedicated to the administrator modules, which included all use cases required for the administrator's role. The third iteration was dedicated to the users' role. Some use cases like posting and update messages were the same apart from the rights granted to each user.

6.2.6 Evaluation

The developer noted a number of lessons emanating from the ICS project. First there was need to revisit the use case diagram at the implementation stage. The update use case was modelled as an <<extend>> use case of post message. In the initial use case, these were stand alone. On implementation the developer noted these could be implemented using the same set of code, with minor adjustment for the update use case.

Section 6.2 has demonstrated the utility of the Secure-SSDM in developing individual sized projects. It should be noted that in this project not all intermediate products were produced during the development process. The suggested intermediate products for this methodology are project specific. The developer should opt for that set of products that ensure maximum productivity for their situation (Cockburn 2004, p.215). Intermediate work products serve for the purposes of project documentation, and in this case the developer should consider both the present and future of the project. For a developer involved in a number of projects, such documentation eases the maintenance of their own systems as they can quickly understand the logic of the system. This applies where there is need for system upgrade or correction in the future. In the event that any other developer is tasked to upgrade the system in the future, they will do so with much ease. Documentation should be sufficient for both current and future purposes. Apart from considering the technical aspect of the documentation, the business side

of the development process should also be addressed. Users should be able to use available documentation to check whether their expectations are being addressed.

6.3 Academic Case study

The first case study to evaluate the utility of the Secure-SSDM was conducted at NUST, Zimbabwe. Undergraduate students studying towards an Honours Degree in Computer Science, took part in the study. The participants were pursuing a four-year degree. Three years of the degree, that is, the first; second and fourth are done in class. The third year is undertaken in industry, where the students are expected to apply the concepts learnt in class in an industry setting. Each academic year is divided into two semesters which are twelve weeks long. This case study was a semester long study.

Thirty-nine second year students took part in the study. These participants were deemed appropriate based on the courses they would have undertaken in their first two years. Students at this level would have done a number of courses in their first and second years which equip them with programming skills, as well as software development skills. These two types of skills are key to the success of this case study. At the end of their second year, students would have done the following courses among others: Systems Analysis and Design; Object-oriented Software Concepts and Development; Software Design Methodologies; Internet and Web Design; and Societal Computing. These five courses are highlighted in this thesis, since they are the most relevant for the study. The following paragraphs elaborate on the content covered in these courses. It should be noted that, the course content described in this thesis is also available from the NUST website. The thick description of the case study environment helps to build "trustworthiness" into this case study (Yin 2015, p.197).

In the course, Systems Analysis and Design (SAD), students cover concepts of structured systems development. These include activities carried out in the stages of the systems development life cycle (SDLC), and the importance of following the stages to facilitate the delivery of quality software. Software development models such as data flow diagrams as process modelling tools and their construction; entity relationship diagrams as data modelling tools and their construction; databases as storage facilities and database definitions are studied. Students also cover concepts in object-oriented analysis and design in the same course. They are taught concepts of object hierarchy and inheritance, and associated concepts. These are important concepts of system modelling, which are important for any participant of this case

study. From SAD, students are expected to undertake a mini project in software development using the structured systems development approach. In undertaking the project, students are expected to apply concepts learnt in their other courses such as Object-Oriented Software Concepts and Development (OOSCD). Such concepts help the students in the coding part of systems development.

In their OOSCD course, students are taught the concepts of software reusability and the use of the Java virtual machine in developing software. Components of the Java virtual machine which include the compiler and interpreter are introduced. Java application programming interfaces are also taught at this level. Apart from the programming skills, students are also taught the importance of software security, and how to build secure software. This research therefore expects that this class of students would have the requisite background needed to participate in this case study. An important background for participating in this study is the knowledge of software development, particularly the concepts of software quality and software security.

The course Software Design Methodologies is a related course to the Systems Analysis and Design course discussed above. It aims at equipping students with software design concepts from various types of software processes. In this course, students are taught how to use software development methodologies. The types of software development methodologies covered include representations from the traditional methods, object-oriented design methods and agile methods. The students are also expected to apply the skills acquired in this course in developing software in a live industry setting. At this point, students have a number of skills to use in developing software. Apart from the programming skills obtained from the OOSCD course, they can use skills from their Internet and Web Design course. In this course the students are taught web programming using tools such as the Hypertext Mark-up Language (HTML) and Cascading Style Sheet (CSS), for example. They also cover Web Content Management using software systems such as Joomla and Drupal. Students also discuss privacy issues for software deployed on the web. These web design skills enable participants to develop web-based systems. In this case study participants were encouraged to develop we-based applications in order to evaluate the utility of the Secure-SSDM. Web-based systems were preferred as they can be developed fast, meaning they can be developed within the semester. Security concerns associated with these applications also make these types of applications ideal for evaluating the security component of the proposed artefact. Participants

were not however restricted to web-based applications only, they were free to develop any type of software to address an identified need or problem in their allocated areas.

The academic case study was undertaken as part of the course, Societal computing. The researcher taught the course in the second semester of the 2018 academic year. This was the third year that the researcher had taught this course. In this course students study design and develop computing applications that address societal needs in areas such as health, education, business, environment (including applications that address climate change) and government among others. Students are also taught the importance of addressing the digital divide when developing computing applications that address societal needs. In this course students also do a semester long project to address a societal need of their choice. In undertaking the project, students are normally encouraged to choose an appropriate methodology as a guide, as well as to choose appropriate platforms for implementing their projects.

These student participants were therefore deemed appropriate as they had the necessary skills acquired from these courses, coupled with skills obtained from other courses covered in their first year. Examples of relevant first-year courses are Software Engineering Concepts, Database Systems and Visual Programming Concepts and Development. These give students a grounding in software development, and make this lot of students ideal as participants to evaluate the methodology. Further, since the participants were not engaged in a major project (as their fourth-year counterparts) at the moment, they had ample time to participate in this study. Importantly, participants were also readily accessible to the researcher.

Before conducting the study, the researcher obtained ethical clearance to undertake the study. Clearance was first obtained from NUST, through the gate keeper. Further, since the research involves humans, an ethical clearance had to be obtained from UNISA. The ethical clearance from UNISA and the clearance from the NUST gate keeper are attached in Appendix A, as Appendix A1 and Appendix A2 respectively. After the clearances were obtained, an invitation to participate in the study was extended to all the second-year students at the beginning of the semester. Thirty-nine students volunteered to participate, out of sixty-nine students. Participants were briefed on the case study and how the results from the study were going to be used.

In the case study, the course was conducted as usual, with participants using the Secure-SSDM in developing software for their term projects. It was made clear to all participants that participation was voluntary and they could choose to pull out at any stage. Pulling out in this

case meant using any other methodology to undertake the project, as it is mandatory for the course. The Secure-SSDM phases and practices in each phase were explained to the participants. After the explanation of the methodology, participants were randomly allocated application areas through a paper raffle. They were made to blindly pick a piece of paper written one of the following: Education; Health; Government; Environment and Business. These are areas normally covered in their Computing in Society course, and would normally constitute areas where they identify their term projects from. Details of the case study are given in the following subsections.

6.3.1 Objectives of the Academic case study

Setting case study objectives helps to focus the case study. The objective of this case study was to evaluate the utility of the Secure-SSDM in developing high-quality and secure software products. The focus of the study was to obtain the perceptions of the solo developers on the effectiveness of the methodology stages and practices embedded in these, in producing the intended impact on the software product. The case study also sought to identify suggestions for improvement from the student participants after they had used the methodology. These would help to refine the methodology.

6.3.2 Case study design

The design of the case study should indicate what is studied (Runeson & Höst 2009, p.139). What is studied in this case is the usability of the Secure-SSDM in designing quality software. Each participant's views are elicited on the effectiveness of the practices embedded in the Secure-SSDM in building quality products. The academic case study was designed as an explanatory case study. An explanatory case study seeks to find causality relationships among concepts in the study (Yin 2015, p.197). At the onset of the study, the methodology, its stages and the associated practices defined in it were explained to the participants. The multiple roles to be played by the developer were elaborated. The user's role in the development was highlighted. Further, tasks and deliverables of each stage of the methodology were explained to the developers.

6.3.3 Case study theory

A theory provides a frame in which a study is conducted. It provides means for disseminating knowledge in a particular environment at the same time supporting decision making from a

practical point of view (Sjøberg et al. 2008, p.313). Ideally a theory consists of constructs, propositions, explanations on why the propositions hold, as well as a defined scope in which the theory holds. The theory used in this case study was derived from existing SSDMs and secure software development methods through a literature review. Table 6.3 summarises the theory on which this case study is based. Quality practices on the first column are expected to enable concrete quality characteristics on the software product as shown on the second column, which in turn enable abstract characteristics on the third column. For example, one proposition from the theory is that if users are educated on the methodology to be used to build the system, they will participate during the development process, and should therefore accept the software product resulting from the development effort. Similarly, the use of small user stories should encourage product understandability which in turn results in a simple product. The case study seeks to ascertain that these propositions hold for the Secure-SSDM as perceived by the solo developers.

Table 6.3: Case study theory

Quality Practices	**Low level characteristics**	**High level Characteristics**
User education Use of small user stories & tasks Refactoring	User participation Product understandability “	User acceptance Product simplicity
Development standards Product validation	Standards adherence “	Consistency
Simple metaphors Test driven development, unit testing Use of a dummy partner Automated code review Task automation Refactoring Version control system	Product simplification Module testability Code quality “ Reusability Product modularity Code traceability	Maintainability
Use of a work break down structure Product backlog	Product comprehension “	Product Completeness
Security awareness training Misuse case detailing Security design principles Security test design Source code security reviews Security testing Review of security repository	Security requirements formulation Misuse case design comprehension Secure designs “ Secure source code “ Security knowledge management	Product security “ “ “ “ “

6.3.4 Data collection

The main methods of data collection with the student participants were class (focus group) discussion and document analysis. The researcher also performed informal observations during practical sessions where students worked on their projects. During these practical sessions, it was also possible to track participants progress and answer questions pertaining to problems they had in applying the methodology.

After the students had used the methodology to develop their systems, a focus group discussion was held with the class to establish the views of the participants on the applicability and effect of the practices as proposed in the case theory. The focus group discussion was held at the end of the semester. A two-hour focus group discussion was conducted to collect the participants' perceptions on the applicability of the practices embedded in the methodology. A teaching assistant attached to the course helped with the data capture of the responses. Before the focus group discussion, the researcher went through the focus group guide and the template prepared to help with the data capture. The teaching assistant helped to capture the data while the researcher moderated the focus group discussion. The moderator also noted key points. The focus group guide used for the session is attached in Appendix C, while the data capture template with sample date for the first question is attached in Appendix D. This research adapted Nili, Tate & Johnstone (2017)'s template for the purpose of systematic data capture. The template was designed to capture ten participants responses per question, since the focus group discussion had a large number of participants, thirty-nine in this case. The responses were captured in the cells of the template designed using Microsoft Excel. After the discussion, the researcher's captured points were synchronised with the teaching assistant notes.

At the end of the projects, participants were also made to submit project documentation though their Google classroom platform. Participants were made to demonstrate their projects to the researcher for the normal evaluation purposes. This is a normal practice for term projects. The software products served to confirm that students had done a project. The system demonstration marks were not included in the analysis. Thirty-five out of thirty-nine submitted documents were analysed. Four participants had incomplete documents, and these were not included in the analysis. Document analysis was done to extract participants comments on the methodology. Participants were asked to include in their methodology section what they perceived as strengths and weaknesses of the methodology. This is normal practice in this

section of the documentation. Students normally consider two or three methodologies and then opt for one based on the argument for the methodology. The researcher also analysed the documents for intermediate artefacts defined in the methodology. The intermediate artefacts analysis was done to confirm the perceptions of the participants on the usability of the Secure-SSDM. Intermediate artefacts analysed include: description of a meeting to educate user, or reasons for not holding the meeting; prioritised product backlog, sprints and sprint backlogs, use cases and misuse cases, design models, test cases, contents of the security repositories and reasons for not having them. The sections of interest from the submitted documents were extracted and entered into a Microsoft Excel spreadsheet for ease of analysis. In the following subsections, results of the data collected in this case study is presented.

6.3.5 Focus group discussion results

In this research editing and template analysis as suggested by Wohlin (2012) are adopted. These are viewed as most appropriate in this thesis as the focus group was conducted using pre-formulated questions.

The responses collected by the researcher and the teaching assistant were synchronised into a single document. Responses were captured per question. While Nili, Tate and Johnstone (2017) suggest that non-verbal data be captured for completeness of data, for this research, verbal data was the main type of data collected, and was the focus of the analysis. It was difficult to capture non-verbal data, except in obvious cases such as show of hands or clapping of the same in agreement. Such expressions were viewed as support for the response at hand. These allowed the researcher to assess popular and non-popular views from the participants regarding the utility of the practices. Table 6.4 shows participants' general comments on the methodology.

Table 6.4: Focus group discussion general comments

Question	Responses
1.Is the Secure-SSDM a solution to a real problem/need in the solo software development environment currently?	-To some extent, when working alone one needs an appropriate guide. -I found it to be filling a gap that exists at the moment -It is to some extent.
2.Would you rate the practices embedded in the methodology adequate to build quality and secure	-Practices are adequate -For me I would add more automated tools to support the processes

software, if not what would you add?	-It is a good methodology to think about security in particular -Add support for code reuse -I would add nothing at the moment
3.How easy to follow are the practices in the Secure-SSDM? Which practices would you consider helpful, and which would you consider to be not?	- The methodology is not that easy to follow -Following a methodology while developing software is not an easy task -A person (classmate) without the knowledge of the language can play the role of a dummy, it helps. -At times users do not have time for meetings -Security design and testing are not easy
4. Did you at any point feel you were asked to do more than just developing software?	-There seems to be a lot of documentation -Models can be used selectively
5.What available tools would you suggest to ease the development process at any of the methodology stages?	-Brackets (free open source front end editing and web development) -Bootstrap eases website development -IBM Watson Assistant API -Node.js, supports both front end and bac end development
6.What practices in the Secure-SSDM would you consider to be key in developing quality and secure software?	-All the practices are necessary, but that should depend on the kind of software -The combined use cases and misuse cases seem to be core in this methodology -The dummy partner was key in my case -Secure coding to me was new, and I feel is key -To me designing test cases seem to serve for the expected quality
7.What improvements would you add to the methodology if you were given the opportunity to?	-Automate most activities -Code reuse -At times users are too busy for user education -User education should only highlight the user's roles
8.Would you consider using the Secure-SSDM in your future projects?	- I would use it on serious projects -Yes, it brings order into the development process -I would, it makes the user think I know what I am doing -I would use it but trim some practices
9.Would you recommend the methodology to any fellow developers?	-I think developers should adopt the methodology, particularly for online applications -I think the Part IVs should consider this on their projects
10. Do you think the Secure-SSDM can be used to develop any kind of software system?	- To some extent -I feel it can be adapted to any environment

Participants agree that the Secure-SSDM is a solution to a real problem. The general perception is that the artefact can be used in solo development environments to build quality software. The practices in the methodology are perceived as important in building quality software. Some developers seem to have reservations on the models to be produced in the

various stages. This reservation was also raised by the industry developers. While developers have these reservations on models, they agree that use case and misuse cases are important in modelling user requirements.

On the part of user education, participants opined that the education should concentrate on highlights of the methodology, particularly on the role of the user in the development process. Table 6.4 also shows that developers would opt to use the methodology in future projects, and they would also recommend the methodology to other developers. They also felt that the methodology could be used to develop any type of software with minimal adjustments.

In the focus group discussion, a phase by phase evaluation of the methodology was made. Table 6.5 shows the responses of the participants. As the table shows, for most phases, participants felt that the practices were adequate. Some participants however felt that in practice it was difficult to adhere to the practices. An example would be a situation where users only have little time to just provide the requirements. Once the requirements are known, they would not avail themselves for some meetings such as initial user education on how development is to proceed, or the recommended sprint review meetings.

Table 6.5: Secure-SSDM phase by phase analysis

Phase	**Participants Responses**
I. Management-buy-in and standards adoption	-Adequate for environment familiarisation -User education should focus on user roles -Important for identifying the users and their roles -Some projects do not have customers you may need other developers to play that role -Standards help to give the developer the non-functional requirements of the system
II. Functional & Security Requirements elicitation	-The practices are adequate for the purpose -Consistent check of requirements with the user helps in building the correct system. -WBS and PBS not necessary for small projects -Misuse cases modelling helps the developer to understand what is expected of them in terms of security.
III. Release & Sprint Planning	-Defined practices are adequate -Security test designs help to deal with the identified security issues -Automated tools ease development practices
IV. Development with code review	-Automate code generation from models -Automated testing -Developing and testing your own code may lead to bias -Dummy partner works to identify errors in code

	-Contract an outsider to perform security testing
V. Sprint Review & Close	-Practices are adequate for the purpose
VI. Evaluation	-Developers may not be genuine in their evaluation of themselves, find someone to do the evaluation -Technical customers can help with the review

The third section of the focus group discussion sought to obtain suggestions for improvement from the participants. Responses were guided by the concepts put across in Table 6.6.

Table 6.6: Suggestions for improvement

Concept	**Participant Responses**
Task adjustment	-Leave methodology and standard to the developer -Automate the development process, in particular make use of code reuse -Get a second developer to review code for critical systems
Provisions for some tasks	Nil
Additions to the methodology	Nil, instead suggested to trim activities
Activities perceived as core	-Requirements elicitation (Use case and misuse cases) -Creating the product backlog -Setting of test cases with user -Continuous integration and testing -User identification and education (on their roles)
Candidate activities for elimination	-User education on methodology -Minimise meetings with customers, they have their own commitments

6.3.6 Focus Group discussion data analysis

Data from the focus group discussion was analysed through identifying key points and assigning codes to these. The coded key points were put into groups. The groups were further organised into themes. Table 6.7 shows the themes emerging from the focus group discussion.

Table 6.7: Focus group data analysis

Theme	**Sub-theme**	**Data Source**
Processes	Practices adequacy	Practices are adequate; All practices are necessary; Need tools to support practices
	Usability	I would use it for serious clients; I would use it but trim some practices; The methodology is not that easy to follow; I feel it can be adapted to any environment
	Time	Users have no time for meetings; Models are time consuming to build; Streamline user education.

	Effectiveness	Solo developers should adopt this methodology; Part IVs should consider this in their projects; Creating product backlog is core
Product	Code quality	Dummy partner is key; Secure coding is key; Test driven development serves for the expected quality;
	Correctness	Combined use cases and misuse cases are core; Consistent check of requirements leads to building a correct system; Misuse cases help to understand product security
	Security	Contract outsider to perform security tests; Misuse cases modelling helps the developer to understand what is expected of them in terms of security; Security test designs help to deal with the identified security issues; Secure coding …is key; Security design and testing are not easy
Developer	Credibility	I would, it makes the user think I know what I am doing; I would use it on serious projects;
	Focus	Standards help the developer to identify non-functional requirements
	Bias	Reviewing of own code might lead to bias; Developers may not be genuine in their evaluation of themselves, find someone to do the evaluation; Get a second developer to review code for critical systems
Users	Education	User education should only highlight the user's roles; At times users are too busy for user education
	Availability	At times users do not have time for meetings; Minimise meetings with customers, they have their own commitments; At times users are too busy for user education; Some projects do not have customers you may need other developers to play that role
	Satisfaction	Early user involvement supports user satisfaction

6.3.7 Document data analysis

In addition to collecting student participants' views from the focus group discussion, students were asked to submit documentation associated with their software products. Thirty five out of thirty-nine students who participated in the case study submitted complete documents for analysis. The other four students submitted incomplete documents; therefore, these were excluded from the analysis. Table 6.8 shows the distribution of software products by area of application as allocated at the onset of the study. The distribution of projects by application area is important as it helps to define a scope of application for the methodology. As shown in Table 6.8 there is a fair distribution of projects among the application areas in the course. Table 6.8 shows that the Secure-SSDM can be used to build software in the areas of education,

health, government, business and the environment. Most of these areas such as health, education, government and business handle sensitive data and would therefore benefit from the security feature of the methodology.

Table 6.8: Project distribution according to application areas

Application area	Number of participants	Percentage (%)
Business	5	14
Education	6	17
Environment	8	23
Government	8	23
Health	8	23
Total	**35**	**100**

To further help with the definition of scope of the methodology, the research analysed the types of software developed by the students. The types were desktop, web-based, mobile applications. While the case study had focused on developing web-based applications, participants were not restricted to these. Table 6.9 shows the types of software products developed by the students. Eighty-eight percent (88%) of the products were web-based with the other types distributed as shown in Table 6.9. The table shows that the Secure-SSDM can be used to develop other types of applications besides web-based applications.

Table 6.9: Types of application systems developed

Type of Application	Count	Percentage
Web-based/ website	31	88%
Mobile-app	2	6%
Desktop	1	3%
Client-server application	1	3%

In the focus group discussion, participants had indicated that the practices in the methodology were adequate to produce a quality software product. Each phase in the Secure-SSDM has associated deliverables, which feed to the next phase. Intermediate deliverables help to guide the developer through the project. The research expects developer participants to produce these as they follow the methodology. Table 6.10 shows the artefacts produced by students in the key phases of the methodology.

Table 6.10: Intermediate models produced by student participants

Type of application	Functional & non-functional requirements analysis models		Design models		Test cases (Quality/Security)	
	Count	**%**	**Count**	**%**	**Count**	**%**
Web-based	28	90	17	55	20	65
Mobile-based	1	50	0	0	0	0
Desktop	1	100	1	100	0	0
Client-server	0	0	0	0	0	0
Total	**30**	**86**	**18**	**51**	**22**	57

Table 6.10 shows that most students (86%) managed to document their requirements using the key models expected. Some students however had the requirements as a listing of the artefacts to be delivered in the methodology. Fewer students produced designs (51%) and test cases (57%) for their systems. Perhaps this confirms some perceptions that the methodology is not easy to follow. While the participants appreciate the importance of the artefacts, they may not be in a position to produce the correct model for the purpose at hand.

In their methodology section of the document, participants were asked to comment on the strengths and weaknesses of the Secure-SSDM. The following were the themes identified from the participants regarding the artefact.

Strengths

- ✓ Accommodation of changes in requirements makes it possible to address user needs which always evolve with time.
- ✓ Frequent customer involvement is a strength for this methodology especially for verifying user needs.
- ✓ The promotion of security makes the methodology favourable for online applications.
- ✓ High transparency of the product under development.
- ✓ Better customer satisfaction due to early user involvement.
- ✓ The use of prototypes enables one to put across a concept to the user with much ease.
- ✓ Development with review enables developers to identify risks early enough.
- ✓ Capturing security requirements early enough gets the developer prepared to tackle security concerns.

- ✓ Misuse cases simplify communicating security concerns to the user.
- ✓ Combining use cases and misuse cases gives the developer an overall picture of the software.
- ✓ This is a low-cost methodology.

Some participants perceive the Secure-SSDM as a low-cost methodology. This addresses the objective of designing a methodology for use in an environment like the solo development environment where resources are limited.

While most participants had given positive feedback on the methodology, some had indicated some negative feedback. Most of the negative feedback was centred around the difficulties associated with incorporating the security aspect into the product. The following list gives negatives noted of the methodology: -

Weaknesses

- × A developer working alone is prone to bias in thinking their ideas are the best.
- × It may be difficult to be honest with some security flaws if a developer is working alone.
- × Security practices are difficult to implement, one might have to contract specialists.
- × It is difficult to identify or engage management in some cases.
- × Minimal documentation makes maintenance by a different person difficult
- × As a freelance developer you may not have some skills, especially those associated with security.
- × Creating some of the intermediate artefacts slows down development process.

The security aspect of the methodology seems to require expertise that may not be available in some developers. In such cases participants recommended the contracting of security experts after carrying out a cost-benefit analysis. Automated tools may also serve for the purpose; therefore, developers are encouraged to spend time researching on what tools exist for their kind of project. Apart from using tools, developers opting for freelance development should consider acquiring some secure software development skills if they are to compete in the software development field.

6.4 Industry Developers Case Study

The industry case study was conducted after the academic case study. The same objectives and theory used for the academic case study were used for the industry case study. At the onset of the industry case study a half-day workshop was held with the participants to explain the methodology to the participants. A sample system for online student registration was modelled to explain the use of the Secure-SSDM. UML diagrams were recommended as modelling tools. The UML diagrams used at each stage for creating models and the tools for implementing the system were dependent on the system under development. A copy of the methodology was handed to each developer for reference purposes.

The objective of the industry case study was to obtain the perceptions of the industry developers on the effectiveness of the Secure-SSDM in building quality and secure software. Data was collected from the participants through interviews. From participant B who had moved to work in another town during the data collection stage, a teleconferencing interview was arranged using the Zoom teleconferencing tool. Zoom was chosen as a it is freely available, particularly for the participant, as there was no budget for participation. Further, Zoom conference participants can share documents among themselves enabling demonstration of ideas diagrammatically (Communications 2019). Two participants were interviewed face to face, each interview lasting an average of one hour. The researcher also kept notes from interactions with participant A who developed the software product detailed in the demonstration section.

In the following sub-sections, the industry case study and the software products developed are overviewed. To maintain anonymity of the developers and the organisations for whom the software was developed codes have been used. The three participants agreed to audio recording. It was easy to capture their perception after the interview.

6.4.1 Participants demographic data

Three participants took part in the industry case study. Participants were made to sign consent forms before the study. The demographic information of the participants is given in Table 6.11. Two male developers and one female developer took part in the study.

Table 6.11: Industry participants demographic data

Participants	Qualification(s)	Gender	Age	Types of Apps Developed	Years of Experience

A	BSc Hons. in Computer Science,	Female	29 Years	Any type, web-based, mobile-based, and desktop applications	4 years
B	BSc Hons. in Computer Science	Male	24 Years	Mobile applications (Android and iOS)	3 years
C	BSc Hons. in Information Technology	Male	30 years	ERP Systems, web-based, mobile and desktop applications	4 years

6.4.2 Case study software projects overview

Participants A and C worked on web-based applications, while participant B developed a mobile application. Participants B and C developed software originated by clients, while A's product was their idea of improving a system their organisation was using. Due to the nature of the project, participant A's project was accessible to the researcher and is used to demonstrate the utility of the Secure-SSDM in Section 6.2. Participant B was performing an upgrade of a mobile application to integrate a payment function. The health application is designed to monitor a patient's medical conditions. Users upload their health readings at time intervals which they set for themselves. In the event that the user forgets to upload readings, the app sends reminders to the user's phone. Upon receiving the readings, the application suggests remedial actions, which include linking the patient with the nearest doctor in their region in case of such a need. Participant C used the Secure-SSDM to develop a system that enhances security to cloud service users by screening IP addresses allowed access to the user's station for sending and receiving messages from the cloud. The details of the two projects were not accessible for ethical reasons.

6.4.3 Results of the industry case study

Interviews were conducted with the three developers after they had used the Secure-SSDM to build individual sized software projects. All the three participants agreed to being audio recorded. Participant B's interview was carried out using Zoom. Each interview was recorded and the themes emanating from thereon captured in a word processor for ease of analysis. This also allowed for member checking as the researcher used this document to confirm the data

captured with the interviewees. The interviewees were asked the same questions using the interview guide, although in some cases follow up questions were asked depending on the situation at hand. The questions in the guide were structured as follows:

Q1. General comments on the methodology

i. Would you consider the Secure-SSDM to be a solution to a real problem/need in the solo software development environment currently?

ii. Would you rate the practices embedded in the methodology adequate to build quality and secure software, if not what would you add?

iii. How easy to follow are the practices in the Secure-SSDM? Which practices would you consider helpful, and which would you consider to be not?

iv. Did you at any point feel you were asked to do more than just developing software?

v. What available tools would you suggest to ease the development process at any of the methodology stages?

vi. What practices in the Secure-SSDM would you consider to be key in developing quality and secure software?

vii. What improvements would you add to the methodology if you were given the opportunity to?

viii. Would you consider using the Secure-SSDM in your future projects?

ix. Would you recommend the methodology to any fellow developers?

x. Do you think the Secure-SSDM can be used to develop any kind of software system?

Table 6.12 shows the responses captured from the participants on general comments.

Table 6.12: General comments by industry participants

Question	**Responses from the participants**		
	Participant A	**Participant B**	**Participant C**
Q1 (i)	The method is a solution to systems that require you to secure the data, i.e. for systems that require the developer to secure data. In some cases,	Secure-SSDM is a solution to a real problem, there is need to develop quality and secure systems	Yes, it covers the whole SDLC, promotes security, & has tools to support the developer

	data security is someone's responsibility		
Q1. (ii)	At times the use case does not really show how the implementation should be, and there may be need to change it during implementation	Misuse cases form the integral part of this methodology, everything stems from use cases and misuse cases	Yes; I would rate the practices 9/10 due to the emphasis on security
Q1. (iii)	Following the practices was not easy, most of the practices were new to me. I think at times creating a use case is confusing as when you get to implementation you may realise there is too much unnecessary information from the use case	Following the methodology first time is a daunting task, but with time it is something doable.	Not easy, prototyping at early stage is not ideal, one might skip some of the security issues, requirements may not be ideal
Q1. (iv)	No suggestions	I suggest the developers to keep standards to themselves, users may not be interested in these	Yes, in building security I had to go for enhanced cryptography, the security part for individuals is too much, perhaps it should be left for consultancy.
Qi. (v)	No suggestions	I can't think of any at the moment	I suggest you include project management tools and cryptography software
Q1. (vi)	The identification of misuse cases is the core function of the method, it imposes the thought of security	Misuse case to me are the highlights of this methodology	The programming approach, use of version control systems, and the use of use cases and misuse cases to capture requirements
Q1. (vii)	At the onset of the project, decide if there is really need for security, before you implement the security feature.	Reduce technical issues to be shared with the user	I can't think of any at the moment

Q1. (viii)	Yes, I would recommend it as it deals with the issue of short cuts, it also gives you a clear view of the project.	I would recommend the Secure-SSDM to other developers	I would rather recommend it to a team, the security feature may be difficult to handle as an individual
Q1. (ix)	I would consider using the methodology in future projects, even for those systems that do not necessarily require security. In such cases I would then strip off the security feature.	I would consider using the methodology in my future projects	Yes, I would
Q1. (x)	Not really, in some cases it might need hybridisation	I would say the methodology can be used to develop any system, but it is mainly suitable for critical systems, in particular the banking environment. It is appropriate in developing the backend of systems that handle client data	Yes, for advanced software development

The three developers agree that the Secure-SSDM is a solution to a real problem. Participant A, however feels that it is more of a solution to those situations where data security is of concern. Two of the three participants perceive use cases and misuse cases as the core practices in developing quality and secure software systems. Participant B noted that applying the misuse case in the case study project had helped them identify a flaw in a system which they would not have identified if they had not used this approach. Misuse cases have been shown to be effective in developing secure software products by a number of authors (Sindre & Opdahl 2005; Belk *et al.* 2011; Robinson & Conkin 2013; Velmourougan *et al.* 2014; Agoda 2016; Ramachandran 2016). Ramachandran (2016, p.583) views misuse cases as a tool for modelling the system requirements from an attacker's perspective; from the author's perspective they form part of best practice in secure software development (p.589).

All the participants felt following the methodology in developing their software products was not easy. This could have been compounded by the fact that from their background information, none of the participants was using any methodology to develop their software systems. However, as seen from participant B's response, given the opportunity to practice

the use of the methodology, using the Secure-SSDM would be something "doable". Participants B and C felt the issue of adopting standards was rather a challenge from a user perspective. Participant C felt the methodology was most appropriate in a team environment where specialist security members would deal with the security part. Leaving the security aspect to a separate security team is a traditional approach to security development. The agile approach to software development empowers the developers to deal with the quality and hence security issues. In the Secure-SSDM since there is only one team member, they have to deal with both aspects of the software.

Participants opined that they would use the Secure-SSDM in their future projects and even recommend the methodology to other developers. Participant C opined that they would rather recommend the methodology to teams, so that the security responsibility is handled by members dedicated to security. From the three participants' perspective the Secure-SSDM can be adapted to develop any kind of software. This perspective is also confirmed by the academic case study where various types of applications were designed by the student participants. Applications developed included web-based applications, mobile-based applications, desktop applications and client-server applications. Participants B and C felt the methodology would be most suitable for critical systems.

Participants were also asked to analyse the methodology phase by phase, pointing out the impact of the practices in each phase on quality. Table 6.13 shows responses the participants gave.

Table 6.13: Industry developers' phase by phase perception of the Secure-SSDM

Phase	Participant A	Participant B	Participant C
I. Management Buy-in & Standards adoption	*Stage I is ok as it is	*Stakeholders are mainly interested in a working system *Standards should be kept to developers for guidance *Solo projects stakeholders do not usually have a budget for quality and security.	*Technical aspect should be hidden from the client.
II. Functional & security requirements elicitation	* Pin users to the original meaning of requirements *Users change requirements meaning without	*Use case and misuse cases are the highlights of the methodology *Misuse cases help the developer to negotiate with the user on the time	*Use case diagrams & associated misuse case are important to communicate

	changing the requirement	required to implement both the use case and misuse case	system functionality
III. Release & Sprint planning	*Practices are adequate	*Misuse cases help to clarify requirements *Time consuming to model use cases *Misuse case may instil fear on the user *Formulate a test for each use case and misuse case. *Too much documentation. *Solo developers hate documentation.	*Include a tool to capture the security issues, this will ease the process of dealing with security requirements
IV. Development with code review	*Get a peer to review your code for critical systems. *Bias in seeing mistakes.	*Aim to make the misuse case to fail the test	*Someone should review your code
V. Sprint review & close	*Ensure users do not change the goal posts. *Users change the meaning of requirements.	No comment	No comment
VI. Evaluation	No comment	No comment	No comment

Asked for suggestions for improvement applicants suggested improvements in the following themes:

- ✓ Keep documentation minimal (Participant B)
- ✓ Standards should be kept to the developer (Participant B)
- ✓ Find a way to keep the developer to their initial meaning of requirements (Participant A)

6.5 Cross Case Study Results Analysis

In analysing the results of the multiple-case study, the research used cross-case analysis. In cross-case analysis, the different cases are analysed separately, after which the results from the component cases are summarised. Data from the individual cases is initially coded, and then analysed to form themes. Codes for the data were derived from the objective of the case study and the literature. In performing cross-case analysis the themes in the component cases

are compared. Similarities and differences are noted. The data are then synchronised to show the overall meaning of data in the multiple case study. The results of the multiple case study are shown in Table 6.14. The table shows the derived themes and sub-themes together with the source of data associated with the sub-themes. For the industry case study, the participants contributing the views are indicated using P.A, P.B, and P.C for participants A, B and C respectively. For the academic case study, the participants contributing views are not cited as the coding was not fixed to a particular participant. However, the table indicates the source of data in the case study. D shows data is sourced from document analysis, while F.G shows the data originates from focus group discussion.

Table 6.14: Cross-case data analysis

Theme	**Sub-theme**	**Academic**	**Industry**
Requirements	Clarity	Frequent customer involvement helps to verify user needs (**D**)	- Misuse cases help to clarify requirements (**P.B**) -Use case diagrams & associated misuse case are important to communicate system functionality (**P.C**)
	Volatility	-Accommodation of changes in requirements makes it possible to address user needs which always evolve with time (**D**).	-Users change requirements meaning without changing the requirement (**P.A**)
Developer	Focus	-Standards help to give the developer non-functional requirements (**F.G**)	-Standards should be kept to developers for guidance (**P.B**)
	Skills	-Developer may not have the necessary skills (**D, FG)**	-The security feature may be difficult to handle as an individual (**P.C**)
	Credibility	-Transparency of the product under development (**D**) -I would, it makes the user think I know what I am doing (**F.G**) -Bias in testing one's code (**F.G; D**)	-It deals with issues of short cuts (**P.A**)
	Peer review	-Get a second developer to review code for critical systems (**F.G; D**)	-Someone should review your code (**P.C**) -Get a peer to review your code for critical systems (**P.A**).

	Time		-Developers do not have time for excessive documentation (**P.B**)
Customer	Availability	-Difficult to engage management (**D**) -Users have no time for meetings (**F.G**; **D)**	
	User education	-Focus on user roles (**F.G**)	-Users are not interested (**P.B**) -Minimise technical issues for users (**P.A; P.C**)
	Satisfaction	-Early user involvement supports user satisfaction (**F.G**)	
Processes	SDLC support	-Practices are adequate for the purpose (**F.G**) -All practices are necessary (**F.G**) -Need for tool support (**F.G**) -The dummy partner is key in code review (**F.G; D**) - Designing test cases serves for the expected quality (**F.G**)	-Covers the whole SDLC (**P.C**) -Practices are adequate (**P.A**) -Use case/misuse cases help the developer to negotiate schedule with client (**P.B)**
	Usability	- I would use it for serious clients (**F.G**) -Ideal for online applications (**D**) -I think the Part IVs should consider this on their projects (**F.G**) -I would use it but trim some practices (**F.G**); -The methodology is not that easy to follow (**F.G**); -I feel it can be adapted to any environment (**F.G**)	-Following the methodology is something doable (**P.B**) -I would consider using the methodology in my future projects (**P.B**) -It gives you a clear view of the project (**P.A**) -I would consider using the methodology in future projects, even for those systems that do not necessarily require security (**P.A**)
	Security	-Misuse cases simplify communicating security concerns to the user (**D**) -Secure coding is key (**F.G**) -Security practices difficult to implement (**D**)	-Promotes security (**P.C**) -I would rate the practices 9/10 due to the emphasis on security (**P.C**) -Misuse cases impose the thought for security (**P.A**)

	Models	-The combined use cases and misuse cases seem to be core in this methodology (**F.G; D**) -Modelling the system through misuse cases helps the developer to understand what is expected of them in terms of security (**F.G; D**) -Prototyping eases user communication (**D**) -Intermediate models slow down development process (**D**)	-Use case/misuse cases help the developer to negotiate schedule with client (**P.B**) -Misuse case instil fear to the customer (**P.B**) -Use case diagrams & associated misuse case are important to communicate system functionality (**P.C)**
	Cost	-Low cost methodology (**D**)	-Minimal budget for quality and security (**P.B**)
Product	Security	-Use of misuse cases (**F.G; D**)	-Use of misuse cases (**P.C**)
	Correctness	-Consistent check of requirements (**F.G**) -Use of a dummy partner (**F.G**)	-Use cases and misuse cases (**P.B; P.C**)

Key: **D**- Document analysis; **F.G**- Focus group discussion; **P.A**- Participant A; **P.B**- Participant B; **P.C**- Participant C

In analysing the multiple case study the research adopts the cross case analysis as suggested by (Cruzes & Dybå 2011). Data from the academic and industry case studies has been analysed separately in Sections 6.4 and 6.5 respectively. In this section data from the two case studies is presented under the broad themes common between the two cases. The broad themes identified in the data are requirements, developer, customer, process and product. The following sub-sections elaborate on these themes.

6.5.1 Requirements

The data from Table 6.14 show how academic and industry participants perceive the Secure-SSDM requirements elicitation process. Results show that both types of participants indicated that the artefact has practices that support requirements clarity. Academic participants perceive frequent customer involvement as promoting requirements clarification. Industry developers perceive combined use cases and misuse cases as promoting requirements clarity.

With regards to requirements volatility, the participants seem to perceive the concept differently. The academic participants perceive support for requirements volatility as a positive aspect of the methodology as it addresses the naturally changing user requirements. On the other hand, industry participants feel at times users unfairly change requirements by changing the meaning of the requirements to conceal requirements changes.

6.5.2 Developer

Regarding support for the developer the following sub-themes emerge from the data; focus, skills, credibility, time and peer review. Participants from both case studies agree that adopting development standards help to focus the developer. However, industry developers feel standards should be reserved to developers and not shared with users. In terms of developer skills, participants feel the lone developer may not have the necessary skills to play both the development and quality control practices. In such cases it is recommended that the developer uses the time they are not running any projects to acquire the necessary skills (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p.29). The results also show that the developer acquires credibility in their projects through the continuous delivery practice since it enhances transparency of the product under development. The industry developers perceive practices in the methodology as dealing with short-cuts that are common with solo developers. On the other hand, participants feel developers could be biased in their code reviews and tests. This could compromise the quality of the product.

To deal with the issue of bias, developers suggested that for critical systems, the developer should get an external peer reviewer. While this is a plausible suggestion, this is dependent on resource availability. To deliver a quality software product, the developer may need to subcontract the security aspect to deal with the security part. Another perception from industry developers is that the methodology requires one to create a number of models which developers may not have time for; “Developers do not have time for excessive documentation (Participant A)”. The models suggested in the Secure-SSDM do not necessarily mean the developer has to design all of them. As observed by one of the academic participants, the models can be applied on demand.

6.5.3 Customer

The perceptions of the developers regarding the customer are divided into the availability, focus, satisfaction and technical issues sub-themes. Academic participants indicated that

customers are difficult to engage in some cases. This makes it difficult to educate users on the development process, or to evaluate the intermediate artefacts with the user. This could have been due to the fact that the projects undertaken in this case were perceived to be for learning purposes by the customers. They may not have seen the benefit of the systems to their organisations. If the developer perceives that the customer is not readily available in a live systems development setting, it is suggested that the developer finds one customer representative interested in the system and works with them during the sprint reviews.

At the same time, participants indicated that early user involvement supports user acceptance. This is an established software engineering practice. When customer consultation takes place at the onset of the project, they buy into the project and are likely to accept the product thereof (González-Sanabria, Morente-Molinera & Castro-Romero 2017; Ramingwong, Ramingwong & Kusalaporn 2017). Regarding customer education, academic developers indicated that education is an important aspect and should focus on the role customers have to play during the development process. Sharing the same views industry participants think education is important, but technical aspects of the project should not be shared with customers.

6.5.4 Secure-SSDM Practices

An important theme arising from the case study is that of the practices embedded in the Secure-SSDM. Participants from both case studies agree that the practices in the methodology are adequate to deliver a high-quality and secure product. Industry participants indicated that the methodology has practices that cover the whole SDLC. The academic case study highlighted the following practices to be key in the development of quality software; requirements elicitation (use case and misuse cases); creating the product backlog; setting of test cases with the user; continuous integration and testing and user identification and education (on their roles). Industry participants also confirmed the importance of the requirements elicitation models. As noted by these participants:

Participant A;

“The identification of misuse cases is the core function of the method; it imposes the thought of security”.

Participant B;

> “Misuse cases to me are the highlights of this methodology”

The quality practices highlighted as core by these participants were also highlighted as so in the quality theory generated in Chapter 2 as promoting quality in the developed software. Further, participants perceive the secure coding practices to be enabling security in the developed software. While participants thought that dealing with security from an individual point of view, was a daunting task, they also felt that the Secure-SSDM was usable with practice. Most participants indicated that they would consider using the methodology especially on critical systems.

Another emerging theme as perceived by these participants is that of cost. Academic participants viewed the methodology as low cost. This addresses the concern raised by one of the industry participants who cited that solo developers' clients usually have no budget for quality and security. Developers adopting this approach may develop quality software at minimal cost. The methodology deals with quality and security issues early in the development process, and thus minimises costs of rework.

6.5.5 Product

The perceptions of both academic and industry developers indicated that products developed using the Secure-SSDM were of high-quality. While participants found the security practices difficult, they acknowledged that their products were secure, as they had to pass the security tests. Industry participants indicated that their products had correct functionality as shown by the acceptance of their users. Participant B's update to incorporate the international payment feature on the mobile application had already been effected, and the product was running as expected. Pointing out the impact of the methodology on the product, participant B had this to say:

"If we had not considered the various ways in which the application could be misused, we would not have noticed the possibility of customers back dating their phones, so as to continue accessing the app services through their phones. Using the Unix atomic clock made it impossible for customers to access services even if they were to back date their phones".

Participants from both case studies opined that frequent checks on customer requirements with the user helped to build a correct product. This is a well-established practice in agile methods. Delivering the product in small increments saves the developer from building a product that the customer may not accept at project end.

6.6 Threats to Validity

This research deals with threats to validity emanating from this case study using the four validity criteria suggested in Runeson et al. (2012). The four criteria are construct, internal, external and reliability.

Construct validity refers to the proper representation of the variables under study by the measures of study. In a multiple case study, construct validity is inherent due to the set-up of the study. In selecting the participants for the academic case study, an open call was made to the class. The research worked with those participants who volunteered to do so. During the course of the study, the researcher monitored intermediate progress to check the application of the methodology by students. This was feasible as the researcher took the class. Students were also asked to submit documentation showing intermediate models towards software product design. The documentation analysis was a means of data triangulation. It also served to confirm that the students were following the practices, so that the results obtained from the project are due to the application of the practices.

Internal validity considers the causal relationships among variables under study. It is important that an investigator establishes that the expected outcomes are due to the factor under investigation and not a third factor (Runeson et al. 2012, p.71). In this study the quality and security of the artefacts under development could come from other sources such as developer experience and component reuse just to name a few. The monitoring of intermediate progress to check the application of the methodology by students served to ensure that the quality of the resultant artefacts was due to methodology practices. This was possible as the researcher took the class therefore prolonging the engagement with the participants. At the data collection stage, a template was prepared in advance to ease the process. A teaching assistant attached to the course helped to collect data from the focus group discussion. This served to ensure that all the data was collected, at the same time serving as a form of researcher triangulation. The focus group discussion held with the academic case study participants also provided an inherent internal validation as participants kept check of each other's perceptions (Nili, Tate & Johnstone 2017).

External validity pertains to the generalisation (or transferability) of the results obtained from a study. A thick description of the academic case study environment provides for the repeatability of the study in a similar environment. The demonstration of the application of the methodology in developing a software product by Participant A also serves for the same

purpose. Since the Secure-SSDM was evaluated both in an academic and industry setting, the results obtained are representative of the two environments.

Reliability of the results is another threat to validity in this study. The multiple case study served to address reliability issues. Further, for the academic case study, since the researcher was an employee of the organisation in which the study was conducted, and had taken the course for the past two years, this means the researcher understood the environment and could easily get access to the data (Runeson et al. 2012, p. 72). Collection of data using more than one method also served to improve data reliability. Member checking was also used in the industry case study to confirm the data collected. Each participant received a Microsoft Word document of their responses against each question asked in the interview through electronic mail. This served to confirm that the researcher had captured their views on the methodology correctly. A feedback meeting at the end of the study also served to confirm that the developers' input was captured as it was.

6.7 Theoretical Evaluation of the Secure-SSDM

The evaluation process in this section is carried out to ensure completeness in the evaluation, at the same time giving the process the necessary rigour. Theoretical evaluation started during the design of the Secure-SSDM. The meta-synthesis carried out in Section 2.5 to build the SSDM quality framework provided a systematic approach to the process of identifying and extracting the quality practices used as a basis for constructing the framework. Using the guidelines from Noblit and Hare (1998), which were refined using recommendations from Sandelowski et al. (1997), practices from SSDMs that met the defined criteria were compared against each other for similarities, differences, and used in the formulation of a line of argument. The identified quality practices were analysed for their capability to build quality software within the context of solo software development. The primary framework created from the meta-synthesis was then presented to academics at a research seminar for their evaluation. Feedback from the computing seminar participants was used to refine the primary SSDM framework. Further, the primary SSDM framework was presented in a blind peer reviewed international conference (Moyo & Mnkandla 2019). Comments from the anonymous reviewers and conference participants were used to refine the framework.

The second phase of rigorous mental evaluation of the Secure-SSDM was conducted during the development of the final artefact. The main aim at this phase was the identification of

security practices compatible with the quality practices in the primary SSDM framework. Besides compatibility, there was also need to evaluate the agility of the practices before and after integration. Section 5.2.2 details this process. The modified version of Keramati and Mirian-Hosseinabadi (2008)'s algorithm provided the guide to systematically bring together the two sets of practices without compromising the agility of the resulting secure-quality practices. This process resulted in the Secure-SSDM. This version of the Secure-SSDM was summarised and submitted for double blind peer review in an international journal. The summary is presented in Moyo and Mnkandla (2020). These formative evaluations served to build quality into the Secure-SSDM. To assess the design objectives set in Chapter 4, a comprehensive evaluation of the artefact using a theoretical model was undertaken. Section 6.7.1 below details this summative theoretical evaluation.

6.7.1 Evaluating the Secure-SSDM using the 4-DAT model

The use of more than one evaluation approach to evaluate a software engineering artefact increases the credibility of the evaluation. Besides the popular experiments and case studies, formal proof in terms of property fulfilment is one of the means of evaluation acceptable in the software engineering domain (Christos 2015, p.5). In this sub-section the final version of the Secure-SSDM is logically evaluated using the four-dimensional analytical tool (4-DAT) (Qumer & Henderson-Sellers 2006; Qumer & Henderson-Sellers 2008). This model was chosen to evaluate the Secure-SSDM's compliance with agile principles. Other researchers (such as Leppa 2013; Ghani, Azham & Jeong 2014; González-Sanabria, Morente-Molinera & Castro-Romero 2017) have also used the model for the same purpose. Using this model, a methodology is assessed according to four dimensions. The four dimensions are method scope, method agility, agile values characterisation and software process characterisation. The 4-DAT framework is a flexible framework. Method evaluators (or researchers) can choose what dimensions to evaluate the methodology against, depending on the purpose of the evaluation. In this thesis, all the four dimensions are applied to give a holistic approach to the evaluation.

Method scope considers the project and team sizes, development environment and coding styles recommended for the method, technology and the physical environments in which the methodology is applicable. It also checks the business and abstraction culture appropriate for the success of the method. Method scope is normally used to perform a high level analysis of a given method (Qumer & Henderson-Sellers 2008, p.281).

Method agility evaluates the existence of five agility features, (i.e. flexibility, speed, leanness, learning and responsiveness) in the method practices and phases. The framework suggests the computation of the degree of agility of each method component as a fraction out of five, based on whether a feature is available (1) or not (0). Degrees of agility can be computed for both practices and phases in a methodology.

Agile values characterisation seeks to identify those components of the methodology which portray the six agile values. The six are: individuals and interactions over processes; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan; keeping the process agile; and keeping the process cost effective (Qumer & Henderson-Sellers 2008, p.281). As can be seen, the first four principles are drawn from the agile manifesto, while the other two principles are the authors' creation to evaluate a methodology's fit for business processes.

Characterisation evaluates a methodology's processes for their ability to support project management as well as process management. Here a methodology is evaluated to check its coverage of the systems development cycle (SDLC). Further, process characterisation checks for the existence of practices focused on project management.

The main aim of this theoretical evaluation was to measure the agility of the methodology using an established model. One of the design goals set in Chapter 4 of this thesis was to design a lightweight agile methodology for use by solo developers. To evaluate the success of this goal, the 4-DAT framework was used, as it focuses on evaluating the agility of a method. The following paragraphs explicate the evaluation processes using the dimensions cited in the framework.

First, the method scope of the Secure-SSDM is considered. This is a high-level evaluation of the method. Table 6.14 shows this analysis performed as suggested by Qumer and Henderson-Sellers (2008). The Secure-SSDM is built to undertake small projects, which can be handled by an individual. Since only one person works in the project, this means the project is normally undertaken in a collocated physical environment. However, this does not stop the developer from developing software for clients across the globe as is the practice nowadays for mobile applications. One of the industry developers in the industry case study worked with an international client to upgrade their mobile-based health application system. Communication in such cases is done online. Development style is iterative and is normally very fast. This is fuelled by the need for survival in the market. The proposed coding style is simple, supported

by technology best familiar to the developer. An object-oriented abstraction mechanism is mainly favoured for use with the methodology. Collaboration with the customer in building the software product is key to a successful project. An object-oriented abstraction supports development velocity. Table 6.15 summarises the Secure-SSDM scope discussed in this paragraph.

Table 6.15: Secure-SSDM method scope evaluation

Method Scope	**Scope evaluation**
Project size	Small
Team size	1
Development style	Incremental and rapid
Code style	Simple
Technology environment	Optional, dependent on developer experience
Physical environment	Collocated
Business culture	Collaborative
Abstraction mechanism	Object oriented, though not restrictive

To evaluate the agility of the Secure-SSDM, this research makes reference to the literature evaluating agile methods. The agility features used for the evaluation are as defined in the 4-DAT framework. In this framework, flexibility measures the ease with which an object or process accommodates emergent changes. This assesses the ability of method processes to respond to changes. Speed refers to the time taken to undertake a process to obtain the expected deliverables. Leanness assesses the general resources used by a process to achieve a desired outcome. In agile methods, lean processes are preferred. Learning assesses the ability of a process to support knowledge management. Knowledge management is an important concept of a solo development environment as it supports developer improvement. Responsiveness measures the process' ability to adapt to the environment (Qumer & Henderson-Sellers 2006, p.504). Table 6.16 shows the assessment of the Secure-SSDM phases based on this framework. The following equations suggested by the authors were used to compute the agility of each of the phases and the overall agility of all the phases:

Agility of a Phase = {Flexibility + Speed + Leanness + Learning + Responsiveness} …… (1)

where the variables in the brackets can either be 0 or 1.

Total Agility of Phases = Agility of {Phase 1 + Phase 2 + ... + Phase n} ……(2)

Total Agility of Processes = $\frac{Total\ agility\ of\ Phases}{n*5}$ ……(3)

Table 6.16: Evaluating the Secure-SSDM phases degrees of agility

Secure-SSDM phase	Agility Features					
	Flexibility	**Speed**	**Leanness**	**Learning**	**Responsiveness**	**Total**
Management Buy-in & Standards adoption	1	1	0	1	1	4/5
Requirements elicitation	1	1	0	1	1	4/5
Release & sprint planning	1	0	0	1	1	3/5
Development with review	1	0	0	1	1	3/5
Sprint review & close	1	1	0	1	1	4/5
Evaluation	1	1	0	1	1	4/5
Total agility	**6/6**	**4/6**	**0/6**	**6/6**	**6/6**	**22/30 =0.73**

Using equation 1 give the agility of the phases as shown in Table 6.16. The table shows that all the phases of the Secure-SSDM exhibit some degree of agility. None of the phases has an agility degree of 5/5 =1. None of the method phases exhibit leanness. This is due to some intermediate documentation necessary in building and tracking of quality concepts in the development process. Other practices violating the leanness feature are requirements elicitation, sprint planning, development review, and evaluation of both the sprint and the project. These practices are associated with some documentation in one way or the other, as models need to be created to explicitly show the processes. However, all the methodology stages have an agility degree higher than 0.5. This characteristic is inherent from the design process used in this research. Only practices with an agile value of more than 0.5 were included in the methodology. Using equation 2 enables the computation of the value 22. Applying equation 3 gives the value: $\frac{22}{6*5} = \frac{22}{30} = 0.73$.

Drawing from the recommendations of Qumer and Henderson-Sellers (2006), the degrees of agility for the practices were derived in a similar manner. Table 6.17 shows the measured degrees of agility of the Secure-SSDM practices. The agility of 0.68 for all the practices compares well with the overall methodology agility of 0.73.

Table 6.17: Evaluating the Secure-SSDM practices degrees of agility

Secure-SSDM Practices	Agility Features					
	Flexibility	**Speed**	**Leanness**	**Learning**	**Responsiveness**	**Total**
i. Education of users on methodology & institution of security awareness programs	1	0	0	1	1	3/5
ii. Adoption of development and security standards	1	1	0	1	1	4/5
iii. Identification of users & user roles	1	1	0	1	1	4/5
iv. Identification of user requirements	1	0	0	1	1	3/5
v. Creation of use cases and misuse cases	1	0	0	1	1	3/5
vi. Creation of a prioritised product backlog	1	0	0	1	1	3/5
vii. Categorisation of subtasks	1	0	0	1	1	3/5
viii. Development of prototypes	1	0	0	1	1	3/5
ix. Use of story cards to explain product characteristics	1	0	0	1	1	3/5
x. Prioritisation of sized sprint tasks	1	1	0	0	1	3/5
xi. Setting of the iteration duration (1 – 2 weeks)	1	1	1	1	1	5/5
xii. Designing of security and acceptance tests	1	0	0	1	1	3/5
xiii. Development of code according to coding and security standards	1	0	0	1	1	3/5
xiv. Use of version/change control tools	1	1	0	1	1	4/5
xv. Performing of code and security code reviews with a dummy partner	1	0	0	1	1	3/5

xvi. Performing unit tests	1	1	1	1	1	5/5
xvii. Review of sprint time & code quality	1	0	0	1	1	3/5
xviii. Movement of finished task (s)	1	1	0	1	1	4/5
xix. Reviewing of project progress	1	0	0	1	1	3/5
xx. Planning for next Sprint (or close project)	1	0	0	1	1	3/5
xxi. Performing of code integration, testing and security testing	1	0	0	1	1	3/5
xxii. Evaluation of product deliverables & security repository update	1	0	0	1	1	3/5
xxiii. Conducting of system acceptance test	1	0	0	1	1	3/5
xxiv. Identification of repeating processes/ tasks for automation	1	1	1	1	1	5/5
Total	**24/24**	**8/24**	**3/24**	**23/24**	**24/24**	**82/ (24*5) = 0.68**

To benchmark the Secure-SSDM practices' degree of agility against existing SSDMs, this research uses agility values computed for DeSoftIn (González-Sanabria, Morente-Molinera & Castro-Romero 2017, p. 31). Table 6.18 shows that the Secure-SSDM's degree of agility is slightly lower than that of DeSoftIn. Both the phases and practices degrees differ by 0.03. It is expected that DeSoftIn would have a higher degree of agility as it does not include security practices which tend to have a negative impact on the agility of a methodology.

Table 6.18: Comparing the agility of the Secure-SSDM to that of DeSoftIn

Methodology	**Agility degree of Phases**	**Agility degree of Practices**
i. DeSoftIn	0.76	0.71
ii. Secure-SSDM	0.73	0.68
Difference	- 0.03	-0.03

In evaluating the Secure-SSDM against the third dimension of the 4-DAT framework of the artefact, the research uses the six agile characteristics as proposed in the framework. These are: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; keeping the process agile; and keeping the process cost effective (Qumer & Henderson-Sellers 2006, p.506). The evaluation process requires that the evaluator indicates practices that support the value under consideration in the method being evaluated. Table 6.19 shows the results of the evaluation. As the table shows, the value, individuals and interactions over processes and tools is supported by these practices in the Secure-SSDM: user education on methodology and security, identification of users and security analysis of user roles, and use of story cards to explain product expectations.

Table 6.19: Evaluating the Secure-SSDM support for agile values

Agile Values	**Practices in the Secure-SSDM supporting the values**
Individuals and interactions over processes and tools	i. User education on methodology & security ii. Identification of users & security analysis of user roles iii. Use of story cards to explain products
Working software over comprehensive documentation	i. 1 – 2 weeks iteration duration ii. Use of prototypes iii. Continuous code integration, testing and security testing
Customer collaboration over contract negotiation	i. Creation of a prioritised product backlog ii. Conducting of system acceptance test
Responding to change over following a plan	i. Reviewing time estimates using actual times ii. Review of sprint time & code quality iii. Adoption of development & security standards iv. Reviewing sprint deliverables with user
Keeping the process agile	i. Sprint review ii. Task automation iii. Review of project progress
Keeping the process cost effective	i. Task automation

Lastly, dimension 4 of the 4-DAT framework is considered. This dimension evaluates the methodology's processes ability to support the product life cycle and the project life cycle. Table 6.20 shows the evaluation of the Secure-SSDM using this dimension. From the table it can be seen that the artefact has practices to support the four specified processes. Most of the

practices however, are concentrated on the development process to enhance developer productivity in this environment where resources are scarce.

Table 6.20: Secure-SSDM characterisation using the 4-DAT Framework

Software Process characterisation	**Practices in the Secure-SSDM supporting the values**
Development process	i. Standards adoption ii. User education iii. Product backlog iv. Sprint v. Continuous integration vi. Testing (quality & security) vii. Code review with dummy
Project management process	i. Release & sprint planning ii. Security repository update
Software configuration control process/Support process	i. Use of version control systems
iv. Process management process	i. Evaluation

The evaluation of the Secure-SSDM based on the four characteristics of the 4-DAT framework is summarised as shown in Figure 6.7. The four quadrants showing the four characteristics indicate that for agile values support and process characterisation there are practices for the purpose. Each of the agile values defined in this framework, has at least one practice in the Secure-SSDM supporting the value. Responding to change over following a plan, has the highest number of practices. Similarly, each process in process characterisation has at least one practice in support of the process. The greatest number of practices (seven in this case) is for the development process.

The quantitative evaluation of the agility of the practices and phases shows that these two comply with agile expectations as defined in the framework. The framework accepts 0.5 as the minimum value for agility. 0.6 is the least agility value or both practices and phases. The overall agility value of the phases is 0.73, while that of practices is 0.68. The method scope of the Secure-SSDM is that of small projects, with team size of one. Development is incremental as is characteristic of agile methods. Development is inherently set for a collocated physical environment. Collaboration with the customer is highly encouraged.

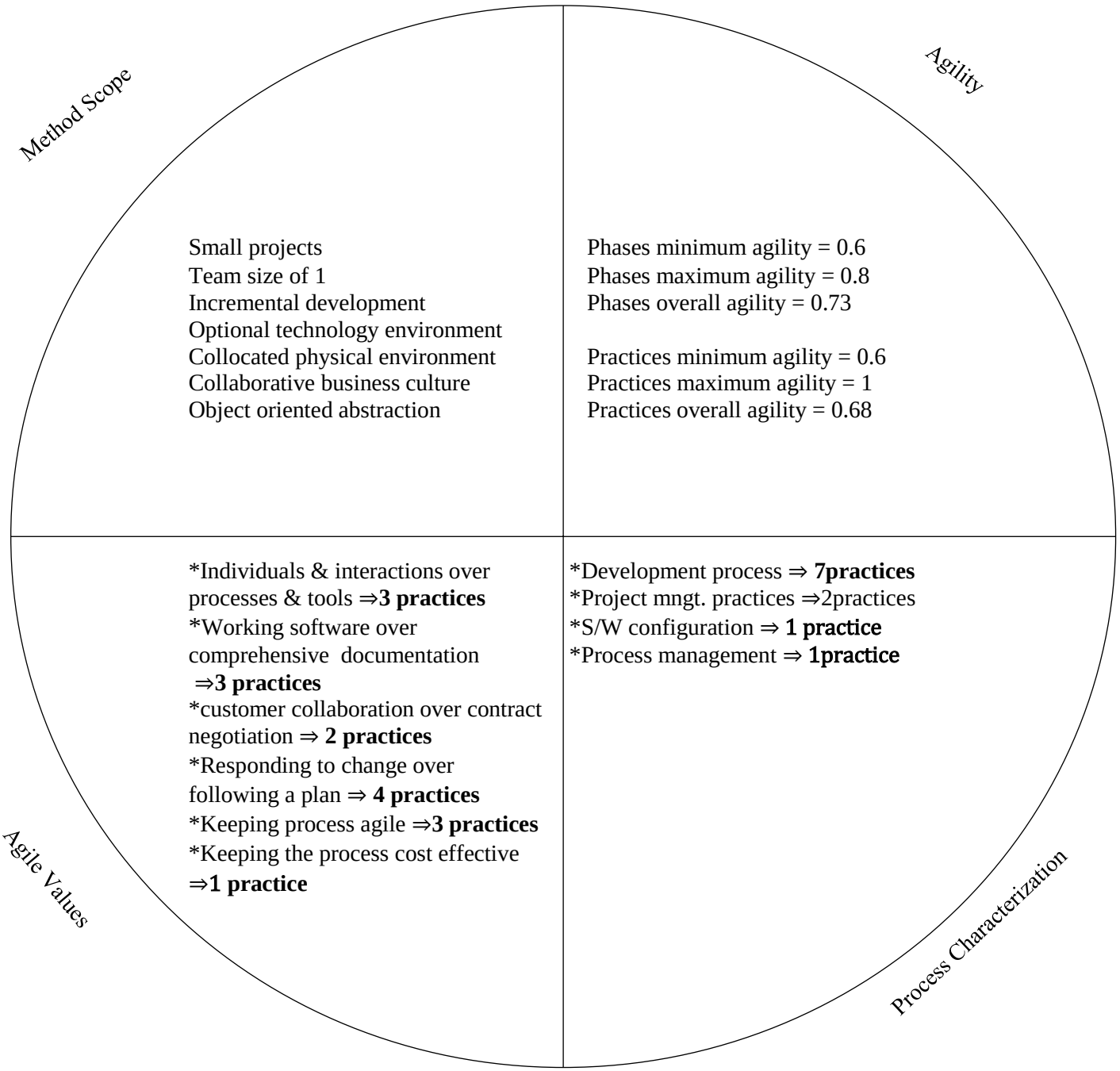


Figure 6.7: Summary of evaluation of the Secure-SSDM using the 4 DAT-Framework

Section 6.7 has demonstrated the theoretical rigour applied during the formative and summative evaluation processes of the Secure-SSDM respectively. In the following sub-section, the results of the theoretical evaluation are discussed.

6.7.2 Theoretical Evaluation Discussion

The evaluation in Sub-section 6.7.1 shows that the Secure-SSDM has all the characteristics expected of an agile methodology. The method scope evaluation indicates that the Secure-SSDM supports small teams, in this case, an individual. Further the methodology is iterative, with fast development speed. It has simple practices, and pursues simplicity where decisions are to be made by users, with regards design decisions. Designed for a team of one, the methodology is inherently collocated, with customer collaborations driving the development process. These characteristics are also seen in agile methods like Scrum and XP (Qumer & Henderson-Sellers 2008, p.284).

Both the practices and stages of the Secure-SSDM have agile degrees greater than 0.5, with the stages of the methodology having an agility degree of 0.73. A value of greater than 0.5 is regarded as agile. In addition, the analysis of the practices embedded in the Secure-SSDM shows that all the agile values are supported by at least one practice in the methodology. The practices: individuals and interactions over processes and tools; working software over comprehensive documentation; responding to change over following a plan; and keeping the process agile are each supported by three or more practices. Customer collaboration over contract negotiation is supported by two practices, while keeping the process cost effective is supported by one practice. This evaluation therefore serves to show that the Secure-SSDM fulfils one of its objective set in Sub-section 4.4.1, that is, the methodology is designed to be a lightweight methodology (agile).

Regarding the Secure-SSDM process characterisation, these practices are embedded in the methodology to ensure life cycle coverage: standards adoption; user education; product backlog formulation; sprint planning; continuous integration; testing (quality & security); and code review with dummy; sprint review. This confirms the perception of one of the industry developers, that the Secure-SSDM has practices covering the complete SDLC. At the maintenance phase the developer can go over the development process as was the case with Participant B who performed an upgrade using the methodology. Further, there is at least one practice each in support of the following: project management process; software configuration control process/ or support process; and process management process.

The discussion in this section has shown that the Secure-SSDM exhibits most of the characteristics defined in the 4-DAT framework as important for an agile methodology to have. The research has therefore achieved to design a solo software development methodology

targeted at individual developers. It can be concluded that the Secure-SSDM is a usable agile method that can be used to design high quality software.

6.8 Chapter Summary

Chapter 6 has given a detailed description of the demonstration and evaluation of the utility of the Secure-SSDM. The demonstration through application of the artefact in Section 6.2 has proved that the methodology practices are usable in designing and implementing quality software products. The perceptions of both academic and industry developers confirm the usability of the Secure-SSDM in building quality products. Developers have confirmed the utility of both the quality and security practices for the purpose.

While some developers raised concerns on the number of models produced in applying the Secure-SSDM, most of the developers applaud the importance of such models. Key among the models is the compound use case and misuse case model. Developers perceive the model to be important in portraying both the functional and security requirements of the product under development. The product backlog was also perceived as an important tool in showing the importance that the user attaches to the product. Another important feature highlighted in this evaluation was the test cases. These were highlighted as improving quality and security of the developed software product.

The theoretical evaluation of the Secure-SSDM has shown that the methodology complies with the expectation of agile methods. Build to support a team of one, the methodology is iterative and delivers the software product incrementally. Both the practices and phases have an agility greater than 0.5, qualifying to be classified as agile (Qumer & Henderson-Sellers 2006, 2008). Further practices embedded in the Secure-SSDM support the four agile values.

CHAPTER 7 CONCLUSION

7.1 Introduction

Chapter 6 presented a demonstration of the application of the Secure-SSDM in designing and implementing high-quality and secure software products. An example software product developed for a university setting was demonstrated. Further the Secure-SSDM was used in a multiple-case study and perceptions of the participants on the utility of the methodology were collected. Study participants concurred on the utility of the artefact in building quality software. A theoretical evaluation of the artefact carried out to enable rigour in the evaluation process, proved that the methodology was compliant with most of the characteristics defined in the model used for the purpose. This chapter presents results and contributions of this thesis. It summarises the answers to the questions posed at the onset of the research and shows outstanding work in the area.

7.2 Answering the Research Questions

The main research question (RQ) that this thesis sought to answer was:

RQ. How can a lightweight solo software development methodology be designed to use as minimum resources as possible, at the same time conforming to the best practice for delivering secure, high-quality software products?

The answer to this main question can be summarised as follows: In defining the lightweight methodology, agile principles as defined in the agile manifesto were adopted. Using Qumer and Henderson-Sellers (2008)'s definition of agility, flexibility, speed, leanness, learning, responsiveness and simplicity were deemed key features for a lightweight methodology. The Secure-SSDM was therefore designed to exhibit these features.

Based on Laporte et al. (2006)'s characterisation of the very small-scale software development environment and review of the existing SSDMs, characteristics of the solo software development environment were derived. These characteristics guided the development of a methodology appropriate for such an environment.

In designing the methodology, quality practices were drawn from solo software development methodologies and related literature, while lightweight secure software development practices were drawn from existing secure software development methods. Based on the proposition by a number of researchers (Keramati & Mirian-Hosseinabadi 2008; Sonia &Singhal 2012;

Ghani, Azham & Jeong 2014), that lightweight quality practices can be integrated with existing traditional security practices without compromising the agility of the resulting practices, Keramati and Mirian-Hosseinabadi's algorithm was adapted for the purpose of integrating the identified practices. Although some researchers have integrated security practices into agile methods designed for teams, this research is unique in that it integrates security practices into a solo development environment.

The summary given in this sub-section is elaborated through the answers provided to the five sub-questions (SQ) posed to help provide an answer to this question. The first sub-question posed was:

SQ1. What methodologies exist for lightweight solo software development?

To answer this question a rigorous literature review was conducted using meta-ethnography as presented in Chapter 2 and revisited in Chapter 4. From this literature review, seven methodologies emerged as leaders in lightweight solo software development. These were Freelance as a Team (Faat); Personal Extreme Programming (PXP1); Personal Extreme Programming (PXP2); Go – Scrum; Scrum Solo; DeSoftIn and MIDS Adaptation.

The identified methodologies have one main focus; to improve the quality of software products, at the same time keeping the process as lightweight as possible, to be undertaken by an individual. This small number of the identified studies confirms the view by a number of authors (Dzhurov, Krasteva & Ilieva 2009; González-Sanabria, Morente-Molinera & Castro-Romero 2017; León-sigg *et al.* 2018) that minimal research exists on SSDMs. While the number is slowly growing, the growth has not fully addressed the improvement of quality in the developed software. Security as a quality feature was found to be missing in the existing SSDMs (Moyo & Mnkandla 2019). These methodologies were however deemed important in this study as they provided this research with a source of quality practices to draw from in order to formulate a higher quality methodology. The pool of methodologies enabled this research to answer the second question formulated as:

SQ2. What software development strategies and techniques in the identified methodologies promote quality in the developed software?

Using meta-ethnography, the quality practices in the identified methodologies were synthesised into a set of themes. The identified practices and themes were as follows:

At software project initiation, the adoption of development standards and user education were found to be key in developing quality software. User education is an established principle in software development. When users are informed about the development process, they participate in the process and will therefore accept the developed product (Ramingwong, Ramingwong & Kusalaporn 2017).

In eliciting user requirements, formulation of user stories using the INVEST (independent, negotiable, valuable, estimable, small and testable) acronym (Bernabé, Navia & García-Peñalvo 2015), design of use cases from these and the subsequent creation of a product backlog to prioritise tasks emerged as key practices to promote requirements understanding and user participation. INVEST is an acronym popularised by Wake (2003), used to assess the quality of user stories. This acronym is now mainly used as an agile guide in most development environments (Lucassen *et al.* 2016; Heck & Zaidman 2018). Use of a prioritised product backlog is also a proven effective way of keeping track of the product components defined in Scrum and is meant to give control of the product under development to the user (Schwaber & Sutherland 2013). In a solo development environment these are key as they give both the developer and the user a complete view of expectations from the project. The same product backlog serves as an input to the planning stage, where sprint tasks are drawn from the list in the order defined by the user. Sprint tasks are recommended to last 1 – 2 weeks.

At the development stage, adoption of test-driven development and the use of a dummy partner to review code were seen to reduce errors and promote code quality. At the same time refactoring complex code, unit testing and the use of version control systems during code implementation were seen to promote product maintainability. Most of these practices are not unique to the solo environment. The use of a dummy partner to review code was however found to be a unique practice for solo development. This replaces the well accepted practice of pair programming. A developer explaining their own code to a dummy partner, is likely to discover errors in code during the process (Bernabé, Navia & García-Peñalvo 2015).

The research also identified continuous integration as a well-established practice in the solo development environment. This is an important agile practice which supports development visibility at the same time supporting frequent delivery of software. In a solo development environment such a practice minimises loss of resources by keeping the customer informed of

project progress. Task automation in this environment was also seen to promote productivity, given the minimal resources available.

SQ3. What lightweight practices and techniques in the software development life cycle promote security in the developed software?

Security practices identified in the literature were organised to fit the various stages of the primary-SSDM derived from the metasynthesis performed to provide the answer in SQ2. Adoption of appropriate security standards and security awareness training were deemed important for initiating a secure software development project. These help the developers to think about security at the onset of the project. Apart from equipping developers with security development skills, these practices were found important in creating security awareness in project stakeholders, so that they participate in identifying security threats in their environment (Microsoft 2008). Rindell, Hyrynsalmi and Leppänen (2018) suggest that in order to keep the practice agile, security items for training can be aligned with the product backlog items.

Practices identified for the planning stage include the use of abuser stories to collect possible system threats and the use of misuse cases to model those threats. Detailing of the misuse case using appropriate models such as sequence diagrams were deemed appropriate for designing a secure system. These show the flow of the unwanted events enabling the solo developer to enact appropriate measures to counter these. Misuse cases can be modelled using UML the same way use cases are modelled. This makes this practice the most favourable in this research as use case modelling is a common practice in software engineering.

Secure source code review was identified as the practice most suitable during system coding. It fitted well with the practice, code review with the help of a dummy partner found in the list of quality practices provided as an answer to SQ2. To enhance productivity, automated code review was deemed most appropriate as a complement to manual code review, for the solo development environment.

SQ4. How can quality and security practices from lightweight software development methodologies be synthesised into a solo software development methodology that promotes quality and security in the developed software?

The answers to the first three questions provided this research with building blocks for use in the design of the Secure-SSDM. These answers were necessary for answering the fourth

question. To answer the fourth question, an algorithm was used to systematically integrate the quality and security practices. The research adapted Keramati and Mirian-Hosseinabadi (2008)'s algorithm for the purpose. The adapted algorithm starts by computing agility degrees for the identified quality and security practices. Once the agility degrees are computed, a compatibility matrix is formulated to determine pairs of practices that can be combined resulting in minimal loss of agility. Before compatible practices are integrated, their resulting degree of agility is tested for optimality. Only those practices with the agility degree greater than 0.5 were integrated. The 0.5 threshold is suggested by Qumer and Henderson-Sellers (2008) as an acceptable measure of agility on a 0 to 1 scale. The integrated practices were then organised into the resulting Secure-SSDM with the following stages: Management-buy-in and standards adoption; Functional and security requirements elicitation; Release and sprint planning; Development with code review; Sprint review and close; Evaluation.

SQ5. How can the resulting methodology be evaluated?

The utility of the Secure-SSDM was empirically evaluated using a multiple case study. One case study was conducted in an academic setting using undergraduate students studying towards a Bachelor of Science honours degree in Computer Science. Participants were asked to use the methodology in designing and implementing their three months mini projects. At the end of the semester participants' perceptions on the usability of the methodology were collected through focus group discussion. Project documents describing how the methodology was used were analysed for the purposes of data triangulation.

For the industry case study three developers in and around Bulawayo, Zimbabwe, participated in the evaluation of the Secure-SSDM. These were identified through a snowballing approach. Developers were asked to use the methodology to develop software products of their choice, preferably for their current clients. Perceptions of these expert developers were collected through face to face interviews and analysed qualitatively.

In evaluating the Secure-SSDM's compliance with agile principles, the 4-DAT framework was used. The framework evaluates a methodology using four agile values, which are: method scope, method agility, agile values characterisation and software process characterisation. In terms of method scope, the Secure-SSDM has been built to support a team of one, uses an iterative approach to software development and delivers the software product incrementally.

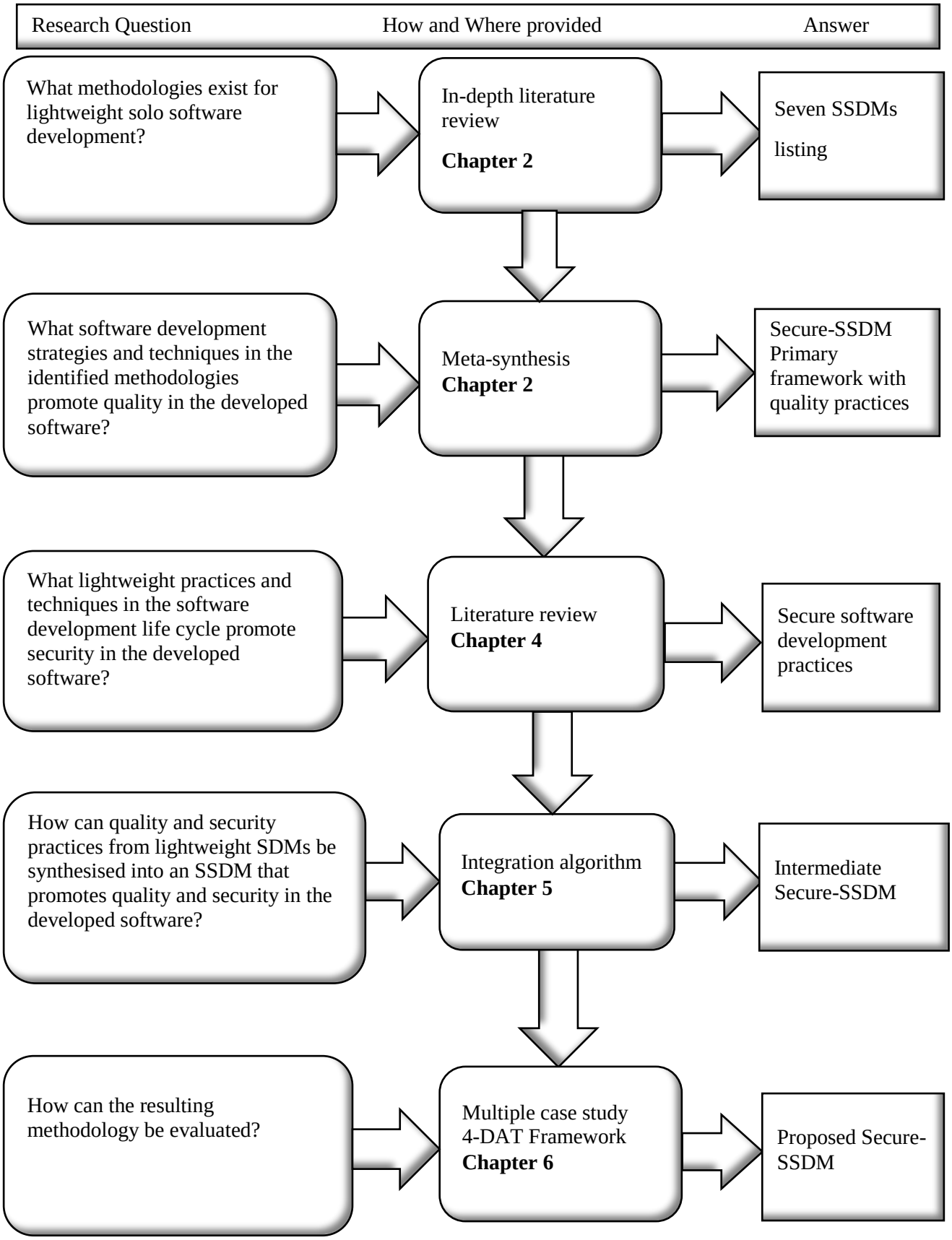


Figure 7.1: Answers to the research questions

An analysis of the Secure-SSDM's method agility has shown that both its practices and phases have an agility degree greater than 0.5, qualifying to be classified as agile (Qumer & Henderson-Sellers 2006, 2008). Agile values characterisation shows that practices embedded in the Secure-SSDM support the four agile values. The answers described in the forgoing paragraphs are summarised in Figure 7.1.

7.3 Unexpected findings from this research

At the evaluation stage of the Secure-SSDM, some surprise findings were noted. The first surprise was the perception by solo developers, that their customers do not value the security of their software products. In interviewing participant B, it was noted that most of the clients they had dealt with were not prepared to invest in the quality and security of their software. The developer related how the client who had approached them for an insurance application, decided to settle for just advertising the insurance premiums, as opposed to advertising and facilitating for payment of the same through the platform. Incorporating the payment platform in this case would mean that security features be incorporated into the platform lengthening the development process at the same time increasing development budget. Participant C concurred on the aspect of security, recommending that security development be left to teams. This participant related how in incorporating security into the platform they had developed in this case study they had to use encryption algorithms which they would not use under normal circumstances. Participants view security coding as a practice for large teams.

The implication of such perceptions is that clients doing business through platforms developed by solo developers remain vulnerable to security threats and possible loss of data and resources. Both solo developers and their customers need to consider secure training as a key aspect in software development.

Another exception in the findings is that some solo developers do not like change in the development process. Participant A passionately shared how disturbing it was for users to continue changing their requirements. Put in their own words:

"At times users change the meaning of the requirements without changing the requirements." They related how they thought that even after adopting such a methodology, and using the use cases in agreeing on certain requirements and modelling these, users would still come and explain the model in a different way. Responding to change as opposed to following a plan is one of the principles in the agile manifesto. Solo developers need to develop a way of

responding to user requests for changes without compromising the quality of their products, and their relationship with customers. A practice recommended in this thesis is the use of prototypes to promote understanding of requirements by all stakeholders in the project.

7.4 Knowledge Contributions

One of the important outputs of DSR is knowledge contribution to the existing knowledge base of the area of study. Throughout the research process, knowledge was generated and contributed to the solo software development environment. In the following sub-sections knowledge contribution in various forms is overviewed.

7.4.1 The Secure-SSDM

The main contribution in this thesis is an artefact in the form of the Secure-SSDM. This research has managed to propose and design a lightweight solo software development methodology with optimum security practices. The security practices had to be optimum to encourage methodology uptake by its intended audience. The Secure-SSDM has been fully documented to show activities, tools and techniques for use at each stage. The utility of the artefact has been evaluated though a multiple case study with developers confirming its usability.

Solo developers can benefit from this methodology, by using it to develop quality and secure products. Given the upsurge in the numbers of solo developers in the software industry, the use of the methodology to develop software by these software developers would also improve the quality of software in the industry.

Researchers on the other hand can improve on the methodology by adding or refining the current quality practices. Further, researchers can perform quantitative evaluation on the defined practices to prove their impact on the designed software products.

7.4.2 Framework of quality practices in the SSD environment.

The second contribution to knowledge was the development of a framework that depicts quality practices in the SSD environment and the outcomes expected when these are applied. This was carried out in Chapter 2. Developers seeking to build quality into their software products can refer to the framework in designing quality software. Researchers intending to

design new methods can build on the framework or refine it as new practices are added to the environment.

7.4.3 Adapted algorithm for integrating quality and security practices.

A third contribution in this research is the adaptation of a quality and security practices integration algorithm for the purpose of using it in a generic environment. This research managed to adapt Keramati and Mirian-Hosseinabadi (2008)'s algorithm designed for use in an organisational setting and applied the resulting algorithm in a generic setting. Researchers wishing to integrate quality practices and security practices in a similar setting, can further adapt the algorithm to suit their purpose.

7.4.4 Research Publications

Communication is one of the stages of DSR. The output of DSR research needs to be packaged and communicated to the intended audience. An important audience during such a research is the academic audience. These serve to prove that the researcher used the right approach in designing the artefact. Two paper publications were made during the course of this thesis. In their chronological order they were:

1. Moyo, S. & Mnkandla, E. (2019) 'A Meta-synthesis of Solo Software Development Methodologies', in *International Multidisciplinary Information Technology and Engineering Conference 2019 (IMITEC 2019)*. Johannesburg.

2. Moyo, S. & Mnkandla, E. (2020) 'A Novel Lightweight Solo Software Development Methodology with Optimum Security Practices', *IEEE Access*.

The third form of communication is this thesis, titled:

3. A Software development methodology for solo software developers: leveraging the product quality of independent developers

7.5 Limitations of the study

The first limitation in this study is that the meta-ethnography process used to develop the primary framework might have missed some SSDM studies that were not published in the electronic sources used. This would mean that some quality practices were not included in the framework. The other limitation is that this research did not define metrics to measure the sub-

characteristics that define the quality characteristics expected of the products designed using this methodology. The utility of the Secure-SSDM in building quality products was only evaluated based on the developers' perceptions. This means the internal quality of the resulting product could not be measured. This was considered out of scope since the research sought to identify and use practices that have been used by other authors for the purpose and therefore practices were assumed to be of the quality claimed.

Another limitation of the study is that a few participants took part in the industry case study. Further, for the industry case study, there were no measures of ascertaining the developers' adherence to the methodology. As a result, the generalisation of the results from the multiple case study is questionable.

7.6 Research Implications

The Secure-SSDM introduces security promoting practices into a solo software development environment where the developer is responsible for both quality and security practices. In secure software development, seperation of duty is a key aspect of security. A developer using the proposed methodology designs, implements and tests both the quality and security of the software product. This gives the developer full control of the software artefact which may compromise the quality and security of the product. Embedding several roles in the same person calls for developers to uphold software development ethics so that they are honest on evaluating the quality and security of their products.

The implication to practice is that solo developers have to be multi-skilled. A solo developer adopting the Secure-SSDM for their software development projects needs to also acquire the security skills besides the quality promoting skills that most developers have. At the implementation stage developers are discouraged from reusing code which they do not understand. While this practice is viewed as increasing developer productivity and is prevalent among solo developers, they have to be willing to create their own code base that will ensure quality and trustworthy software code.

From a busines point of view the results of this study give solo developers a competitive advantage in satisfying their clients. Haq et al. (2018) indicate that security is the highest-ranking satisfaction factor that clients look for in web-based applications ahead of ease of use, user interface and information. Therefore, solo developers adopting practices embedded in

the Secure-SSDM stand to improve the quality and security of the software products resulting in improved client satisfaction rates which may in turn lead to improved client following.

While freelance software development has been considered as an alternative cheap source of software products, as indicated by one of the industry participants in Section 7.3, the pricing gap may be reduced. Clients of solo developers may need to be prepared to absorb the cost that comes with secure software development. To fully benefit from the proposed methodology, software champions need to be willing to provide a budget commensurate with the security expected from the software product.

7.7 Recommendations for further work

This research provided answers for most of the questions asked as detailed in Section 7.2. However, the research did not perform an internal evaluation of the quality and security of the products designed by the developers. In this research, this was assumed to be inherent based on the practices used to design the Secure-SSDM. The perceptions of the developers were used for the purpose. Further research can build on this research by conducting controlled experiments to evaluate the quality of products built using the methodology.

There are other quality practices that were shown to be missing in the quality framework when compared with the IEE/IEC 25010 quality model. These include product characteristics portability and efficiency. Further research can be conducted to introduce quality practices that support these characteristics.

REFERENCES

Abrahamsson, P. *et al.* (2002) 'Improving Software Developer's Competence: Is the Personal Software Process Working?', in *Empirical Software Engineering*. Rovaniemi, Finland, pp. 1–8. Available at: http://arxiv.org/abs/1311.0228. (Accessed: 18 August 2017)

Abrantes, J. F. & Travassos, G. H. (2011) 'Common Agile Practices in Software Processes', in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, pp. 355–358. doi: 10.1109/ESEM.2011.47.

Agarwal, R. & Umphress, D. (2008) 'Extreme programming for a single person team', in *46th Annual Southeast Regional Conference on XX*. Auburn, AL, USA.: ACM, pp. 82–87. Available at: http://dl.acm.org/citation.cfm?id=1593105.1593127. (Accessed: 27 August 2017)

Aguda, O. A. (2016) *Effectiveness of Security Requirements Engineering in Agile/Scrum Software Development Projects*. Colorado Technical University.

Ahmed, M. A. & Hoven, J. van den (2010) 'Agents of responsibility-freelance web developers in web applications development', *Information Systems Frontiers*, 12(4), pp. 415–424. doi: 10.1007/s10796-009-9201-0.

Al-amin, S. *et al.* (2018) 'Toward effective adoption of secure software development practices', *Simulation Modelling Practice and Theory*, 85, pp. 33–46. doi: https://doi.org/10.1016/j.simpat.2018.03.006.

Al-Tarawneh, M. Y., Abdullah, M. S. & Ali, A. B. M. (2011) 'A proposed methodology for establishing software process development improvement for small software development firms', *Procedia Computer Science*. Elsevier, 3, pp. 893–897. doi: 10.1016/j.procs.2010.12.146.

Albadarneh, A., Albadarneh, I. & Qusef, A. (2015) 'Risk management in Agile software development: A comparative study', *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pp. 1–6. doi: 10.1109/AEECT.2015.7360573.

Alexander, I. (2003) 'Misuse Cases : Use Cases with Hostile Intent', *IEEE Software*, pp. 58–66.

Almomani, M. A. *et al.* (2016) ‘An Empirical Analysis of Software Practices in Malaysian Small and Medium Enterprises’, in *3rd International Conference On Computer And Information Sciences (ICCOINS)*. Kuala Lumpar, Malaysia: IEEE, pp. 442–447.

Ambler, S. W. (2001) ‘Determining What to Build: Object-Oriented Analysis’, in *The Object Primer*. 2nd edn. Cambridge, United Kingdom: Cambridge University Press, pp. 181–240.

Amjad, S. *et al.* (2017) ‘Calculating Completeness of Agile Scope in Scaled Agile Development’, *IEEE Access*, 6, pp. 5822–5847. doi: 10.1109/ACCESS.2017.2765351.

Anacona, D. *et al.* (2015) ‘Innova-Procedure : A procedure to guide the innovation of software development processes in VSEs Innova-Procedure : Un procedimiento para guiar la innovación de procesos de desarrollo software en VSEs’, pp. 108–115.

Atlassian (2019) *Trello*. Available at: https://trello.com (Accessed: 25 September 2019).

Ayalew, T., Kidane, T. & Carlsson, B. (2013) ‘Identification and Evaluation of Security Activities in Agile Projects’, in H., N. R. and Gollmann, D. (eds) *NordSec 2013*. Berlin: Springer-Verlag Berlin Heidelberg, pp. 139–153.

Ayalew, Y. & Motlhala, K. (2014) ‘Software Process Practices in Small Software Companies in Botswana’, in *2014 14th International Conference on Computational Science and Its Applications*. IEEE Computer Society Press, pp. 49–57. doi: 10.1109/ICCSA.2014.20.

Baca, D. *et al.* (2015) ‘A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting’, in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, pp. 11–19. doi: 10.1109/ARES.2015.45.

Baca, D. & Carlsson, B. (2011) ‘Agile development with security engineering activities’, in *Proc. 2011 Intl. Conference on Software and Systems Process*. Honolulu: ACM, pp. 149–158. doi: 10.1145/1987875.1987900.

Barafort, B. *et al.* (2018) ‘A software artefact to support standard-based process assessment : Evolution of the TIPA ® framework in a design science research project’, *Computer Standards & Interfaces*. Elsevier, 60(February), pp. 37–47. doi: 10.1016/j.csi.2018.04.009.

Barbara, K. & Pfleeger Lawrence, S. (1996) ‘Software quality: The Elusive target’, *IEEE Software*, 13(1), pp. 12–21.

Baskerville, R. *et al.* (2018) ‘Design Science Research Contributions : Finding a Balance

between Artifact and Theory', *Journal of the Association for Information Systems*, 19(5), pp. 358–376. doi: 10.17705/1jais.00495.

Basri, S. Bin & O'Connor, R. V. (2010) 'Organizational Commitment towards Software Process Improvement: An Irish Software VSEs Case Study', in *Proceedings 2010 International Symposium on Information Technology - System Development and Application and Knowledge Society, ITSim'10*, pp. 1456–1461. doi: 10.1109/ITSIM.2010.5561489.

Beck, K. (2000) *Extreme Programming Explained: Embrace Change, Addison-Wesley*. Boston, MA, USA: Addison - Wesley. doi: 10.1136/adc.2005.076794.

Beck, K. *et al.* (2001) 'Manifesto for Agile Software Development', pp. 2–3. Available at: https://www.researchgate.net/file.PostFileLoader.html?id=57d055b593553b11467ddd59&assetKey=AS%3A403742915612673%401473271220194. (Accessed 20 September 2017)

Beck, K. & Andres, C. (2004) *Extreme Programming Explained: Embrace Change*. 2nd edn. Adison Wesley.

Belk, M. *et al.* (2011) *Fundamental Practices for Secure Software Development*. 2nd edn. Edited by S. Simpson. Software Assurance Forum for Excellence in Code (SAFECode).

Bernabé, R. B. ., Navia, Á. . & García-Peñalvo, J. . (2015) 'Faat - Freelance as a Team', in *Third International Conference on Technological Ecosystems for Enhancing Multiculturality-TEEM '15*. Porto, Portugal: ACM, pp. 687–694. doi: http://dx.doi.org/10/1145/2808580.2808685.

Beznosov, K. & Kruchten, P. (2004) 'Towards Agile Security Assurance', in *New Security Paradigms Workshop 2004*. White Point Beach Resort, Nova Scotia, Canada: ACM, pp. 47–54.

Boehm, B. (2006) 'A view of 20th and 21st century software engineering', in *Proceedings of the 28th International Conference on Software Engineering SE - ICSE '06*. Shanghai: ACM, pp. 12–29. doi: doi: 10.1145/1134285.1134288.

Boehm, B. & Turner, R. (2009) *Balancing Agility and Discipline, A Guide for the Perplexed*. 7 th. Boston: Pearson Education, Inc.

Brereton, P. *et al.* (2007) ‘Lessons from applying the systematic literature review process within the software engineering domain’, *Journal of Systems and Software*. Elsevier Inc., 80(4), pp. 571–583. doi: 10.1016/j.jss.2006.07.009.

Cahill, M. *et al.* (2018) ‘Qualitative synthesis: A guide to conducting a meta-ethnography’, *British Journal of Occupational Therapy*, 81(3), pp. 129–137. doi: 10.1177/0308022617745016.

Calvo-Manzano, J. A. *et al.* (2012) ‘Methodology for process improvement through basic components and focusing on the resistance to change’, *Journal of software: Evolution and Process*, pp. 511–523.

Christos, K. (2015) ‘A Taxonomy of Evaluation Approaches in Software Engineering’, in *BCI’15*. Craiova, Romania: ACM, pp. 1–8. doi: http://dx.doi.org/10.1145/2801081.2801084.

Cockburn, A. (2004) *Crystal Clear : a human-powered methodology for small teams*. Pearson Education, Inc.

Coleman, G. & O’Connor, R. V. (2008) ‘An investigation into software development process formation in software start-ups’, *Journal of Systems and Software*, 81(5), pp. 772–778. doi: 10.1108/17410390810911221.

Communications, Z. V. (2019) *Zoom Pricing*. Available at: https://zoom.us (Accessed: 10 October 2019).

Crispin, L. (2006) ‘Driving Software Quality : How Test-Driven Development Impacts Software Quality’, *Software, IEEE*, 23(6), pp. 70–71. doi: 10.1109/MS.2006.157.

Cruzes, D. S. & Dybå, T. (2011) ‘Research synthesis in software engineering: A tertiary study’, *Information and Software Technology*, 53(5), pp. 440–455. doi: 10.1016/j.infsof.2011.01.004.

Davis, N. (2013) *Secure Software Development Life Cycle Processes*. Pittsburgh. Available at: http://resources.sei.cmu.edu/asset_files/whitepaper/2013_019_001_297287.pdf. (Assessed: 28 May 2018)

Dent, A. (2008) ‘From Scrum to Solo: How Small is Too Small a Team to Still Call it Software Engineering?’, in Aitken, A. and Rosbotham, S. (eds) *19th Australian Software*

Engineering Conference:ASWEC 2008. Barton, ACT:Engineers, Australia, pp. 152–159. Available at: http://www.aswec2008.curtin.edu.au/IndustryReport/Dent - SCRUM to Solo.pdf. (Accessed: 24 August 2017)

Diaz, A. *et al.* (2016) 'The ISO/IEC 29110 Implementation on two Very Small Software Development Companies in Lima. Lessons Learned', *IEEE Latin America Transactions*, 14(5), pp. 2504–2510. doi: 10.1109/TLA.2016.7530452.

Dingsøyr, T. *et al.* (2018) 'Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation', *Empirical Software Engineering*. Empirical Software Engineering, 23(1), pp. 490–520. doi: 10.1007/s10664-017-9524-2.

Dingsøyr, T., Faegri Erland, T. & Itkonen, J. (2014) 'What is Large in Large-Scale ? A Taxonomy of Scale for Agile Software', in Jedlitschka, A. et al. (eds) *Product-Focused Software Process Improvement*. Springer International Publishing, pp. 1–5. doi: 10.1007/978-3-319-13835-0.

Dittrich, Y. (2016) 'What does it mean to use a method ? Towards a practice theory for software engineering', *Information and Software Technology*. Elsevier B.V., 70, pp. 220–231. doi: 10.1016/j.infsof.2015.07.001.

Doyle, M. *et al.* (2014) 'Agile Software Development in Practice', in *XP 2014*. Switzerland: Springer International Publishing, pp. 32–45. doi: 10.4018/978-1-59904-927-4.ch001.

Driessen, V. (2010) *A Successful Git Branching Model, 05 January*. Available at: https://nvie.com/posts/a-successful-git-branching-model/ (Accessed: 16 November 2018).

Driessen, V. (2018) *Git power tools for daily use, 08 November*. Available at: https://nvie.com/posts/git-power-tools/ (Accessed: 16 November 2018).

Dybå, T. & Dingsøyr, T. (2008) 'Empirical studies of agile software development: A systematic review', *Information and Software Technology*, 50(9–10), pp. 833–859. doi: 10.1016/j.infsof.2008.01.006.

Dzhurov, Y., Krasteva, I. & Ilieva, S. (2009) 'Personal Extreme Programming–An Agile Process for Autonomous Developers', in *International Conference SOFTWARE, SERVICES & SEMANTIC TECHNOLOGIES (S3T)*. Sofia, Bulgaria, pp. 252–259.

Easterbrook, S. *et al.* (2008) 'Selecting Empirical Methods for Software Engineering Research', in Shull, F., Singer, J., and Sjøberg, D. I. K. (eds) *Guide to Advanced Empirical Software Engineering*. Springer, pp. 285–311.

Eclipse Foundation (2018) 'Eclipse Process Framework: EPF 1.5.2 Release'. Available at: https://www.eclipse.org/epf/ (Accessed: 10 August 2018).

Elvesæter, B., Benguria, G. & Ilieva, S. (2013) 'A comparison of the Essence 1.0 and SPEM 2.0 specifications for software engineering methods', *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering - PMDE '13*, pp. 1–10. doi: 10.1145/2489833.2489835.

Fayad, M. E., Laitinen, M. & Ward, R. P. (2000) 'Software Engineering in the Small', *Communications of the ACM*, 43(3), pp. 115–118. Available at: http://portal.acm.org/citation.cfm?doid=330534.330555. (Accessed: 18 October 2017)

Fioravanti, F. (2011) 'eXtreme Programming', *Skills for Managing Rapidly Changing IT Projects*, (February), pp. 108–133. doi: 10.4018/978-1-59140-757-7.ch009.

Firdaus, A., Ghani, I. & Jeong, S. R. (2014) 'Secure Feature Driven Development (SFDD) Model for Secure Software Development', in *International Conference on Innovation, Management and Technology Research*. Malaysia: Elsevier B.V., pp. 546–553. doi: 10.1016/j.sbspro.2014.03.712.

Fitzgerald, B. & Stol, K. J. (2017) 'Continuous software engineering: A roadmap and agenda', *Journal of Systems and Software*. Elsevier Ltd., 123, pp. 176–189.

Fowler, M. & Highsmith, J. (2001) *The Agile Manifesto*.

Fuggetta, A. (2000) 'Software Process : A Roadmap', in *Future of Software Engineering*. Limerick: ACM, pp. 1–12. doi: 10.1145/336512.336521.

Galvan, S. *et al.* (2015) 'A Compliance Analysis of Agile Methodologies with the ISO/IEC 29110 Project Management Process', *Procedia Computer Science*. Elsevier Masson SAS, 64, pp. 188–195.

García-Mireles, G. A. *et al.* (2012) 'Towards the harmonization of process and product oriented software quality approaches', *Communications in Computer and Information Science*, 301 CCIS, pp. 133–144. doi: 10.1007/978-3-642-31199-4_12.

García-Mireles, G. A. *et al.* (2015) 'Approaches to promote product quality within software process improvement initiatives: A mapping study', *Journal of Systems and Software*, 103, pp. 150–166.

Garg, K. (2017) *Case Study Oriented Learning Environment for Software Engineering*. International Institute of Information Technology, Hyderabad, India.

Ghani, I., Azham, Z. & Jeong, S. R. (2014) 'Integrating software security into agile-Scrum method', *KSII Transactions on Internet and Information Systems*, 8(2), pp. 646–663. doi: 10.3837/tiis.2014.02.019.

Gherib, B., Baghdadi, Y. & Kraiem, N. (2015) 'A method engineering perspective for service-oriented system engineering', *International Journal of Web Information Systems*, 11(4), pp. 62–99. doi: 10.1108/IJWIS-03-2015-0004.

González-Sanabria, J. S., Morente-Molinera, J. A. & Castro-Romero, A. (2017) 'DeSoftIn : A methodological proposal for individual software development', *Revista Facultad de Ingeniería (Rev. Fac. Ing.)*. Pedagogical and Technological University of Colombia (UPTC), 26(45), pp. 23–32. doi: http://doi.org/10.19053/01211129.v26n44.2017.5768.

Gregor, S. (2006) 'The Nature of Theory in Information Systems', *MIS Quarterly*, 30(3), pp. 611–642.

Hakim, H., Sellami, A. & Abdallah, H. Ben (2016) 'Evaluating Security in Web Application Design Using Functional And Structural Size Measurements', in *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, pp. 182–190. doi: 10.1109/IWSM-Mensura.2016.16.

Hap (2020) *Hireaprogrammer*. Available at: https://www.hireaprogrammer.co.za/ (Accessed: 26 August 2020).

Haq, N. U. *et al.* (2018) 'Determinants of client satisfaction in web development projects from freelance marketplaces', *International Journal of Managing Projects in Business*, 11(3), pp. 583–607. doi: 10.1108/IJMPB-02-2017-0017.

Hayes, W. & Over, J. (1997) *The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers, CMU/SEI-97-TR001*.

Heck, P. & Zaidman, A. (2018) 'A systematic literature review on quality criteria for agile requirements specifications', *Software Quality Journal*. Springer US, 26, pp. 127–160. doi: 10.1007/s11219-016-9336-4.

Hevner, A. *et al.* (2004) 'Design Science Research in Information Systems', *MIS quarterly*, 28(1), pp. 75–105. doi: 10.2307/25148625.

Hevner, A. R. *et al.* (2004) 'Design Science in Information Systems Research', *MIS Quarterly*, 28(1), pp. 75–105. doi: 10.2307/25148625.

Hollar, A. B. (2006) *Cowboy: An Agile Programming Methodology for a Solo Programmer*. Virginia Commonwealth University.

Homaei, H. & Shahriari, H. R. (2019) 'Athena : A framework to automatically generate security test oracle via extracting policies from source code and intended software behaviour', *Information and Software Technology*, 107, pp. 112–124.

Howard, M., LeBlanc, D. & Viega, J. (2010) *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw Hill.

Hsieh, J. & Hsieh, Y. (2013) 'Appealing to Internet-based freelance developers in smartphone application marketplaces', *International Journal of Information Management*. Elsevier Ltd, 33(2), pp. 308–317. doi: 10.1016/j.ijinfomgt.2012.11.010.

Hughes, B. & Cotterrell, M. (2012) *Software Project Management*. 5th edn. Berkshire, England: McGraw- Hill Higher Education.

Humphrey, W. S. (1995) *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co Inc.

Humphrey, W. S. (2000) *The Personal Software Process (PSP) The Personal Software Process SM (PSP SM), TECHNICAL REPORT CMU/SEI-2000-TR-022 ESC-TR-2000-022*.

Idri, A. *et al.* (2017) 'ISO/IEC 25010 Based Evaluation of Free Mobile Personal Health Records for Pregnancy Monitoring', *Proceedings - International Computer Software and Applications Conference*, 1, pp. 262–267. doi: 10.1109/COMPSAC.2017.159.

IEEE Computer Society (2014) *IEEE Standard for Software Quality Assurance Processes, IEEE Std 730-2014 (Revision of IEEE Std 730-2002).* doi:

10.1109/IEEESTD.2014.6835311.

Iqbal, J. *et al.* (2016) 'Software SMEs' unofficial readiness for CMMI®-based software process improvement', *Software Quality Journal*. Springer US, 24(4), pp. 997–1023. doi: 10.1007/s11219-015-9277-3.

ISO/IEC (2014) *TECHNICAL REPORT ISO / IEC TR 29110-5-6-2 Systems and software engineering — Lifecycle profiles for Very Small and engineering guide : Generic profile group : Basic profile*. Switzerland.

ISO/IEC (2018) 'INTERNATIONAL STANDARD ISO / IEC Information technology — Security techniques — Information security management systems — Overview and'. Switzerland: ISO/IEC, pp. 1–27.

ISO (2010) *ISO/IEC FCD 25010: Systems and software engineering–system and software product quality requirements and evaluation(SQauRE)–System and software quality models.*

Janus, A. *et al.* (2012) 'The 3C approach for Agile Quality Assurance', in *3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*. Zurich, Switzerland: IEEE, pp. 9–13.

Kabbedijk, J. & Jansen, S. (2011) 'Steering insight: An exploration of the Ruby software ecosystem', *Lecture Notes in Business Information Processing*, 80 LNBIP, pp. 44–55.

Kadi, I., Idri, A. & Ouhbi, S. (2016) 'Quality evaluation of cardiac decision support systems using ISO 25010 standard', *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, (1), pp. 1–8. doi: 10.1109/AICCSA.2016.7945657.

Karim, N. S. A. *et al.* (2016) 'The practice of secure software development in SDLC: an investigation through existing model and a case study', *Security and Communication Networks*, 9(18), pp. 5333–5345. doi: 10.1002/sec.1700.

Keramati, H. & Mirian-Hosseinabadi, S.-H. (2008) 'Integrating Software Development Security Activities with Agile Methodologies', in *2008 IEEE/ACS International Conference on Computer Systems and Applications*. Doha: IEEE, pp. 749–754. doi: 10.1109/AICCSA.2008.4493611.

Keshta, N. & Morgan, Y. (2017) 'Comparison between traditional plan-based and agile

software processes according to team size & project domain (A systematic literature review)', in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, pp. 567–575. doi: 10.1109/IEMCON.2017.8117128.

Kruchten, P. (2002) *A Software Development Process for a Team of One*. Available at: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/feb02/ProcessForOneFeb02.pdf (Accessed: 14 August 2017).

Laporte, C., Alexandre, S. & O'Connor, R. (2008) 'A Software Engineering Lifecycle Standard for Very Small Enterprises', in Springer-Verlag (ed.) *Software Process Improvement*. Berlin, Heidelberg, pp. 129–141. doi: 10.1007/978-3-540-85936-9_12.

Laporte, C., April, A. & Renault, A. (2006) 'Applying ISO / IEC Software Engineering Standards in Small Settings : Historical Perspectives and Initial Achievements', in *Proceedings of SPICE 2006 Conference*. Luxembourg, pp. 1–6.

Laporte, C., Renault, A. & Alexandre, S. (2008) 'The Application of International Software Engineering Standards in Very Small Enterprises', in Oktaba, H. and Piattini, M. (eds) *Software Process Improvement for Small and Medium Enterprises : Techniques and Case Studies*. IGI Global, pp. 42–70.

Laporte, C. Y. *et al.* (2017) 'Systems engineering and management processes for small organizations with ISO/IEC 29110: An implementation in a small public transportation company', in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, pp. 1–8. doi: 10.1109/SYSCON.2017.7934718.

Larrucea, X. *et al.* (2016) 'Software Process Improvement in Very Small Organizations', *IEEE Software*, 33(April), pp. 85–89. doi: 10.1109/MS.2016.42.

Laukkanen, E., Itkonen, J. & Lassenius, C. (2017) 'Problems, causes and solutions when adopting continuous delivery—A systematic literature review', *Information and Software Technology*. Elsevier B.V., 82, pp. 55–79. doi: 10.1016/j.infsof.2016.10.001.

León-sigg, M. De *et al.* (2018) 'Adaptation of the Initial Software Development Method for a Single Developer', in *6th International Conference in Software Engineering Research and Innovation*. San Luis Potosi, Mexico: IEEE, pp. 35–41. doi: 10.1109/CONISOFT.2018.00013.

Leppa, M. (2013) ‘A Comparative Analysis of Agile Maturity Models’, in Pooley, R. (ed.) *Information Systems Development:Reflections, Challenges and New Directions*. New York: Springer Science + Business Media, pp. 329–343. doi: 10.1007/978-1-4614-4951-5.

Lew, P. (2012) ‘An Enterprise Framework for Evaluating and Improving Software Quality’, in *PNSQC 2012*. PNSQC.ORG, pp. 1–11.

Lucassen, G. *et al.* (2016) ‘Improving agile requirements: the Quality User Story framework and tool’, *Requirements Engineering*. Springer London, 21(3), pp. 383–403. doi: 10.1007/s00766-016-0250-x.

Lucia, A. De & Qusef, A. (2010) ‘Development Requirements Engineering in Agile Software Development’, *Journal of Emerging Technologies in Web Intelligence*, 2(3), pp. 212–220. doi: 10.4304/jetwi.2.3.212-220.

Magdaleno, A. M., Werner, C. M. L. & Araujo, R. M. De (2012) ‘Reconciling software development models: A quasi-systematic review’, *Journal of Systems and Software*, 85(2), pp. 351–369.

Malik, U. M., Nasir, H. M. & Javed, A. (2014) ‘An Efficient Objective Quality Model for Agile Application Development’, *International Journal of Computer Applications*, 85(8), pp. 975–8887.

Marchewka, J. T. (2015) *Information Technology Project Management: providing measurable organisational value.* 5th edn. Hobken: John Wiley and Sons.

Maxim, B. R. & Kessentini, M. (2016) ‘An introduction to modern software quality assurance’, in *Software Quality Assurance*. Elsevier Inc., pp. 19–46. Available at: http://dx.doi.org/10.1016/B978-0-12-802301-3.00002-8.

McCall, J. a., Richards, P. K. & Walters, G. F. (1977) ‘Factors in software quality: Concept and Definitions of Software Quality’, I(November), p. 188.

McGraw, G. (2005) *The 7 Touchpoints of Secure Software*.

Microsoft (2008) ‘MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (SDL)’. Microsoft Corporation, pp. 1–78.

Mnkandla, E. (2016) *A META-SYNTHESIS ON THE USABILITY OF SOCIAL MEDIA BLENDS IN E-LEARNING, University of South Africa*. Univeristy of South Africa.

Mohammad, A., Alqatawna, J. & Abushariah, M. (2017) ‘Secure software engineering: Evaluation of emerging trends’, in *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*, pp. 814–818. doi: 10.1109/ICITECH.2017.8079952.

Mohammed, M. A., Moles, R. J. & Chen, T. F. (2016) ‘Meta-synthesis of qualitative research: the challenges and opportunities’, *International Journal of Clinical Pharmacy*. Springer International Publishing, 38(3), pp. 695–704. doi: 10.1007/s11096-016-0289-2.

Moyo, S. & Mnkandla, E. (2019) ‘A Metasynthesis of Solo Software Development Methodologies’, in *International Multidisciplinary Information Technology and Engineering Conference 2019 (IMITEC 2019)*. Johannesburg.

Moyo, S. & Mnkandla, E. (2020) ‘A Novel Lightweight Solo Software Development Methodology with Optimum Security Practices’, *IEEE Access*, 8, pp. 33735–33747. doi: 10.1109/ACCESS.2020.297100.

Mtsweni, J. S. (2013) *iSEMSERV: A FRAMEWORK FOR ENGINEERING INTELLIGENT SEMANTIC SERVICES*. University of South Africa.

Napoleão, B. M. & Rodrigo, P. (2018) ‘Using meta-ethnography to synthesize research on knowledge management and agile software development methodology’, in *Brazil Symposium on Software Quality (SBQS), October 17–19*. Curtiba, Brazil: ACM. doi: https://doi.org/10.1145/3275245.3275270.

Naur, P. & Randell, B. (1968) *Software Engineering, Report on a conference sponsored by the NATO SCIENCE COMMITTEE*. doi: 10.1111/j.1432-1033.1992.tb16798.x.

Nili, A., Tate, M. & Johnstone, D. (2017) ‘A framework and approach for analysis of focus group data in information systems research’, *Communications of the Association for Information Systems*, 40(Article 1), pp. 1–21. doi: 10.17705/1cais.04001.

Nistala, P. V. *et al.* (2016) ‘Quality management and Software Product Quality Engineering’, in *Software Quality Assurance*. Elsevier Inc., pp. 133–150. doi: 10.1016/B978-0-12-802301-3.00006-5.

Noblit, G. W. & Hare, R. D. (1998) ‘Meta-Ethnography : Synthesizing Qualitative Studies’, in *Particularities: Collected Essays on Ethnography and Education*. Peter Lang AG, pp. 93–123.

Nurdiani, I. *et al.* (2019) ‘Understanding the order of agile practice introduction : Comparing agile maturity models and practitioners ’ experience’, *The Journal of Systems & Software*. Elsevier Inc., 156, pp. 1–20. doi: 10.1016/j.jss.2019.05.035.

Nwasra, N., Basir, N. & Marhusin, M. F. (2016) ‘A framework for evaluating QinU based on ISO/IEC 25010 and 25012 standards’, in *2015 9th Malaysian Software Engineering Conference, MySEC 2015*, pp. 70–75. doi: 10.1109/MySEC.2015.7475198.

O’Connor, R. V. & Laporte, C. Y. (2014) ‘An Innovative Approach to the Development of an International Software Process Lifecycle Standard for Very Small Entities’, *International Journal of Information Technologies and Systems Approach*, 7(1), pp. 1–22. doi: 10.4018/ijitsa.2014010101.

Oates, B. J. (2006) *Researching Information Systems and Computing*. London: SAGE Publications Ltd.

Object Management Group Inc. (2011) *Business Process Model and Notation (BPMN)*. Available at: http://www.omg.org/spec/BPMN/2.0. (Accessed: 18 May 2018)

Osterweil, L. J. (1997) ‘Software processes are software too, revisited’, *Proceedings of the 19th international conference on Software engineering ICSE 97*, Boston, Ma, pp. 540–548. doi: 10.1145/253228.253440.

Othmane, L. *et al.* (2014) ‘Extending the Agile Development Process to Develop Acceptably Secure Software’, *IEEE Transactions on Dependable and Secure Computing*. IEEE, 11(6), pp. 497–509. doi: 10.1109/TDSC.2014.2298011.

Oueslati, H., Rahman-Masudar, M. & Othmane-ben, L. (2015) ‘Literature Review of the Challenges of Developing Secure Software Using the Agile Approach’, in *10th International Conference on Availability, Reliability and Security*. Toulouse, France: IEEE, pp. 540–547. doi: 10.1109/ARES.2015.69.

OWASP (2006) *OWASP CLASP*. Available at: https://owasp.org/ (Accessed: 26 December 2019).

OWASP (2017) *Software Assurance Maturity Model*. Mountain View, USA.

Pagotto, T. *et al.* (2016a) ‘Scrum Solo’, in *11th Iberian Conference on Information Systems and Technologies (CISTI)*. Las Palmas, Spain: IEEE, pp. 1–6. doi:

10.1109/CISTI.2016.752155.

Pagotto, T. *et al.* (2016b) ‘Scrum Solo’, in *I1th Iberian Conference on Information Systems and Technologies (CISTI)*. Las Palmas, Spain: IEEE, pp. 1–6. doi: 10.1109/CCISTI.2016.752155.

Palsetia, N. *et al.* (2016) ‘Securing native XML database-driven web applications from XQuery injection vulnerabilities’, *The Journal of Systems & Software*. Elsevier Inc., 122, pp. 93–109. doi: 10.1016/j.jss.2016.08.094.

Pardo, C. *et al.* (2011) ‘Harmonizing quality assurance processes and product characteristics’, *Computer*, 44(6), pp. 94–96. doi: 10.1109/MC.2011.178.

Paternoster, N. *et al.* (2014) ‘Software Development in Startup Companies : A Systematic Mapping Study’, *Information and Software Technology*, 56(20), pp. 1200–1218.

Pedreira, O. *et al.* (2007) ‘A systematic review of software process tailoring’, *ACM SIGSOFT Software Engineering Notes*, 32(3), pp. 1–6. doi: 10.1145/1241572.1241584.

Peffers, K. *et al.* (2008) ‘A Design Science Research Methodology for Information Systems Research’, *Journal of Managenet Information Systems*, 24(3), pp. 45–77. doi: 10.2753/MIS0742-1222240302.

Pohl, C. & Hof, H.-J. (2015) *Secure Scrum: Development of Secure Software with Scrum*. doi: 10.1109/UBMK.2017.8093383.

Porres, I. *et al.* (2013) ‘Authoring IEC 61508 Based Software Development Process Models’, in *14th International Conference on Product-Focused Software Process Improvement (PROFES 2013)*. Berlin Heidelberg: Springer-Verlag, pp. 268–282. doi: 10.1007/978-3-642-39259-7_22.

Pressman, R. & Maxim, B. (2015a) *Software Engineering: A Practitioner’s approach.* 8th edn. New York: McGraw Hill.

Pressman, R. & Maxim, B. (2015b) *Software Engineering: A Practitioner’s Approach*. 8th edn. New York, New York, USA: McGraw Hill,Education.

Qumer, A. & Henderson-Sellers, B. (2006) ‘Measuring Agility and Adoptability of Agile Methods : A 4-Dimensional Analytical Tool’, in *IADIS International Conference Applied Computing*. IADIS Press, pp. 503–507.

Qumer, A. & Henderson-Sellers, B. (2008) ‘An evaluation of the degree of agility in six agile methods and its applicability for method engineering’, *Information and Software Technology*, 50(4), pp. 280–295. doi: 10.1016/j.infsof.2007.02.002.

Rafi, U. *et al.* (2015) ‘US-Scrum: A Methodology for Developing Software with Enhanced Correctness, Usability and Security’, *International Journal of Scientific & Engineering Research*, 6(9), pp. 377–383. Available at: http://www.ijser.org. (Accessed: 20 August 2019)

Rafique, Y., Mišić, V. B. & Misic, V. B. (2013) ‘The effects of test-driven development on external quality and productivity: A meta-analysis’, *IEEE Transactions on Software Engineering*, 39(6), pp. 835–856. doi: 10.1109/TSE.2012.28.

Ragunath, P. *et al.* (2010) ‘Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC)’, *International Journal of Computer Science and Network Security*, 10(1), pp. 112–119.

Ramachandran, M. (2016) ‘Software security requirements management as an emerging cloud computing service’, *International Journal of Information Management*, 36, pp. 580–590.

Ramingwong, L., Ramingwong, S. & Kusalaporn, P. (2017) ‘Solo Scrum in Bureaucratic Organization: A Case Study from Thailand’, in Kim, K. J., Kim, H., and Baek, N. (eds) *IT Convergence and Security, 2017*. Gateway East, Singapore: Springer Verlag, pp. 341–348.

Raunak, M. S. & Binkley, D. (2017) ‘Agile and other trends in software engineering’, in *2017 IEEE 28th Annual Software Technology Conference (STC)*. Gaithersburg, MD, USA: IEEE, pp. 1–7. doi: 10.1109/STC.2017.8234457.

Raymund, S. *et al.* (2005) ‘Personal Software Process (PSP) Assistant’, in *Software Engineering Conference*. Taipei, Taiwan: IEEE, pp. 687–694.

Richardson, I. & Gresse von Wangenheim, C. (2007) ‘Guest Editors’ Introduction: Why are Small Software Organizations Different?’, *IEEE Software*, 24(1), pp. 18–22. doi: 10.1109/MS.2007.12.

Rindell, K., Hyrynsalmi, S. & Leppänen, V. (2017) ‘Busting a Myth :Review of Agile Security Engineering Methods’, in *Availability, Reliability and Security - ARES ’17*. Reggia Calabria, Italy: ACM, pp. 1–10. doi: 10.1145/3098954.3103170.

Rindell, K., Hyrynsalmi, S. & Leppänen, V. (2018) 'Aligning Security Objectives With Agile Software Development', in *XP '18 Companion*. Porto, Portugal: ACM, New York, NY,USA, pp. 1–9.

Robinson, G. & Conkin, L. (2013) *Code review guide 2.0*.

Runeson, P. *et al.* (2012) *CASE STUDY RESEARCH IN SOFTWARE ENGINEERING*. 1 st. New York, USA: John Wiley and Sons.

Runeson, P. & Höst, M. (2009) 'Guidelines for conducting and reporting case study research in software engineering', *Empirical Software Engineering*, 14(2), pp. 131–164. doi: 10.1007/s10664-008-9102-8.

Sandelowski, M., Docherty, S. & Emden, C. (1997) 'Focus on qualitative methods. Qualitative metasynthesis: issues and techniques.', *Research in nursing & health*, 20(4), pp. 365–371. doi: 10.1002/(SICI)1098-240X(199708)20:4<365::AID-NUR9>3.0.CO;2-E.

Santos, R. E. S., Magalhães, C. V. C. & da Silva, F. Q. B. (2017) 'Member Checking in Software Engineering Research : Lessons Learned from an Industrial Case Study', in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement Member*. IEEE Computer Society, pp. 187–192. doi: 10.1109/ESEM.2017.29.

Schwaber, K. (1997) 'SCRUM Development Process', in *Business Object Design and Implementation*. London: Springer, pp. 117–134. Available at: http://link.springer.com/10.1007/978-1-4471-0947-1_11. (Accessed: 12 July 2018)

Schwaber, K. & Sutherland, J. (2013) *The Scrum Guide, Scrum.Org and ScrumInc*. doi: 10.1053/j.jrn.2009.08.012.

Schwalbe, K. (2012) *Information Technology Management*. 7 th. Boston: Course Technology, Cengage Learning.

Seaman, C. B. (1999) 'Qualitative Methods in Empirical Studies of Software Engineering', *IEEE Transactions on Software Engineering*, 25(4), pp. 557–572.

Selleri Silva, F. *et al.* (2015) 'Using CMMI together with agile software development: A systematic review', *Information and Software Technology*. Elsevier B.V., 58, pp. 20–43.

Sfetsos, P. *et al.* (2016) 'Integrating user-centered design practices into agile Web

development: A case study', in *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pp. 1–6. doi: 10.1109/IISA.2016.7785424.

Sfetsos, P. & Stamelos, I. (2010) 'Empirical studies on quality in agile practices: A systematic literature review', in *Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010*. IEEE, pp. 44–53. doi: 10.1109/QUATIC.2010.17.

Siau, K. & Long, Y. (2005) 'Synthesizing e-government stage models – a meta-synthesis based on meta-ethnography approach', *Industrial Management & Data Systems*, 105(4), pp. 443–458. doi: 10.1108/02635570510592352.

Sillitti, A. and Succi, G. (2006) 'Requirements Engineering for Agile Methods', *Engineering and MaNanaging Software Requirements*.

Sindre, G. & Opdahl, A. L. (2005) 'Eliciting security requirements with misuse cases', *Requirements Engineering*, 10(1), pp. 34–44. doi: 10.1007/s00766-004-0194-4.

Sjøberg, D. I. K. *et al.* (2008) 'Building Theories in Software Engineering', in Shull, F., Singer, J., and Sjøberg, D. I. K. (eds) *Guide to Advanced Empirical Software Engineering*. London: Springer-Verlag, pp. 312–320. doi: 10.1007/978-1-84800-044-5.

Solyman, A. M., Ibrahim, O. A. & Elhag, A. A. M. (2015) 'Project management and software quality control method for small and medium enterprise', in *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, pp. 123–128. doi: 10.1109/ICCNEEE.2015.7381442.

Sommerville, I. (2011) *Software Engineering*. 9th edn. Pearson Education, Inc.

Sonia & Singhal, A. (2012) 'Integration Analysis of Security Activities from the perspective of agility', in *Agile India*. Bengaluru, India: IEEE Computer Society, pp. 40–47. doi: 10.1109/AgileIndia.2012.9.

Sonia, Singhal, A. & Banati, H. (2014) 'Fisa-Xp', *ACM SIGSOFT Software Engineering Notes*, 39(3), pp. 1–14. doi: 10.1145/2597716.2597728.

Steiner, D. (2015) *The Best Freelancer Websites for Finding Developers, Business.com*.

Stewart, J. *et al.* (2012) 'A qualitative metasynthesis of activity theory in SIGDOC proceedings 2001-2011', in *Proceedings of the 30th ACM international conference on*

Design of communication - SIGDOC '12, p. 341. doi: 10.1145/2379057.2379120.

Suryn, W. (2014) 'Software Quality Engineering: Making it Happen', in *Software Quality Engineering: A practioner's Approach*, pp. 35–138. doi: 10.1002/9781118830208.

Suteeca, K. & Ramingwong, S. (2017) 'A framework to apply ISO/IEC29110 on SCRUM', *20th International Computer Science and Engineering Conference: Smart Ubiquitos Computing and Knowledge, ICSEC 2016*. doi: 10.1109/ICSEC.2016.7859884.

Sutton, S. M. (2000) 'The Role of Process in a Software Start-up', *IEEE Software*, 17(4), pp. 33–39.Available at: https://pdfs.semanticscholar.org/0e53/13231a32191482ca484d80d6a51a95ec7b54.pdf.

Trienekens, J., Kusters, R. & Van Solingen, R. (2002) 'Product Focused Software Process Improvement: Concepts and Experiences from Industry', *Software Quality Journal*, 9(4), pp. 269–281. doi: 10.1023/A:1013715203889.

Uikey, N. (2015) 'A Lifecycle Model for Web-based Application Development: Incorporating Agile and Plan-driven Methodology', *International Journal of Computer Applications*, 117(19), pp. 28–36. doi: 10.5120/20664-3400.

Vaishnavi, V., Kuechler, B. & Petter, S. (2017) *DESIGN S CIENCE R ESEARCH IN INFORMATION SYSTEMS, Association for Information Systems*. Available at: http://www.desrist.org/design-research-in-information-systems (Accessed: 20 June 2019).

Velmourougan, S. *et al.* (2014) 'Software development Life cycle model to build software applications with Usability', in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 271–276.

Venable, J., Pries-Heje, J. & Baskerville, R. (2016) 'FEDS: A Framework for Evaluation in Design Science Research', *European Journal of Information Systems*, 25(1), pp. 77–89. doi: 10.1057/ejis.2014.36.

Viega, J. (2005) *The clasp application security process, Training Manual*. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+CLASP+Application+Security+Process#0.

Wagner, S. *et al.* (2015) 'Operationalised product quality models and assessment: The Quamoco approach', *Information and Software Technology*. Elsevier B.V., 62(1), pp. 101–

123. doi: 10.1016/j.infsof.2015.02.009.

Wahyuni, D. (2012) 'The Research Design Maze: Understanding Paradigms, Cases, Methods and Methodologies', *Journal of applied management accounting research*, 10(1), pp. 69–80.

Wake, B. (2003) *INVEST in Good Stories, and SMART Tasks*. Available at: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/ (Accessed: 27 June 2019).

Walls, J. G., Widmeyer, G. R. & Sawy, O. A. El (1992) 'Building an Information System Design Theory for Vigilant EIS', *Information Systems Research*, 3(1), pp. 36–59.

Wäyrynen, J., Bodén, M. & Boström, G. (2004) 'Security Engineering and eXtreme Programming: An Impossible Marriage?', in *Conference on Extreme Programming and Agile Methods*. Springer-Verlag, pp. 117–128. doi: 10.1007/978-3-540-27777-4_12.

Wesslén, A. (2000) 'A Replicated Empirical Study of the Impact of the Methodsin the PSP on Individual Engineers', *Empirical Software Engineering*, 5(2), p. 93. Available at: http://portal.acm.org/citation.cfm?id=594384.594473.

Wieringa, R. & Daneva, M. (2015) 'Six strategies for generalizing software engineering theories', *Science of Computer Programming*. Elsevier B.V., 101, pp. 136–152. doi: 10.1016/j.scico.2014.11.013.

Wohlin, C. (2012) 'Case Studies', in *Experimentation in software engineering*. Springer-Verlag Berlin Heidelberg. doi: 10.1007/978-3-642-29044-2.

Wongsai, N., Siddoo, V. & Wetprasit, R. (2015) 'Factors of influence in software process improvement: An ISO/IEC 29110 for very-small entities', in *Proceedings - 2015 7th International Conference on Information Technology and Electrical Engineering: Envisioning the Trend of Computer, Information and Engineering, ICITEE 2015*, pp. 12–17. doi: 10.1109/ICITEED.2015.7408904.

Yin, R. K. (2015) *Case Studies*. Second Edi, *International Encyclopedia of the Social & Behavioral Sciences: Second Edition*. Second Edi. Elsevier. doi: 10.1016/B978-0-08-097086-8.10507-0.

Zarour, M. *et al.* (2015) 'An investigation into the best practices for the successful design and

implementation of lightweight software process assessment methods: A systematic literature review', *Journal of Systems and Software*. Elsevier Ltd., 101, pp. 180–192. doi: 10.1016/j.jss.2014.11.041.

APPENDICES

Appendix A1 UNISA Ethical Clearance

UNISA

UNISA COLLEGE OF SCIENCE, ENGINEERING AND TECHNOLOGY'S (CSET) RESEARCH AND ETHICS COMMITTEE

23 November 2018

Ref #: 078/SM/2018/CSET_SOC
Name: Mrs Sibonile Moyo
Student #: 61514780
Staff #:

Dear Mrs Sibonile Moyo

Decision: Ethics Approval for 5 years (Humans Involved)

Researchers: Mrs Sibonile Moyo, National University of Science and Technology, Computer Science Department, Corner Gwanda Road, P. Bag AC 939, Ascot, Bulawayo, Zimbabwe, 61514780@mylife.unisa.ac.za, +263 947 0239, +263 77 235 1126

Project Leader(s): Prof E Mnkandla, mnkane@unisa.ac.za, +27 11 670 9059

Working title of Research:

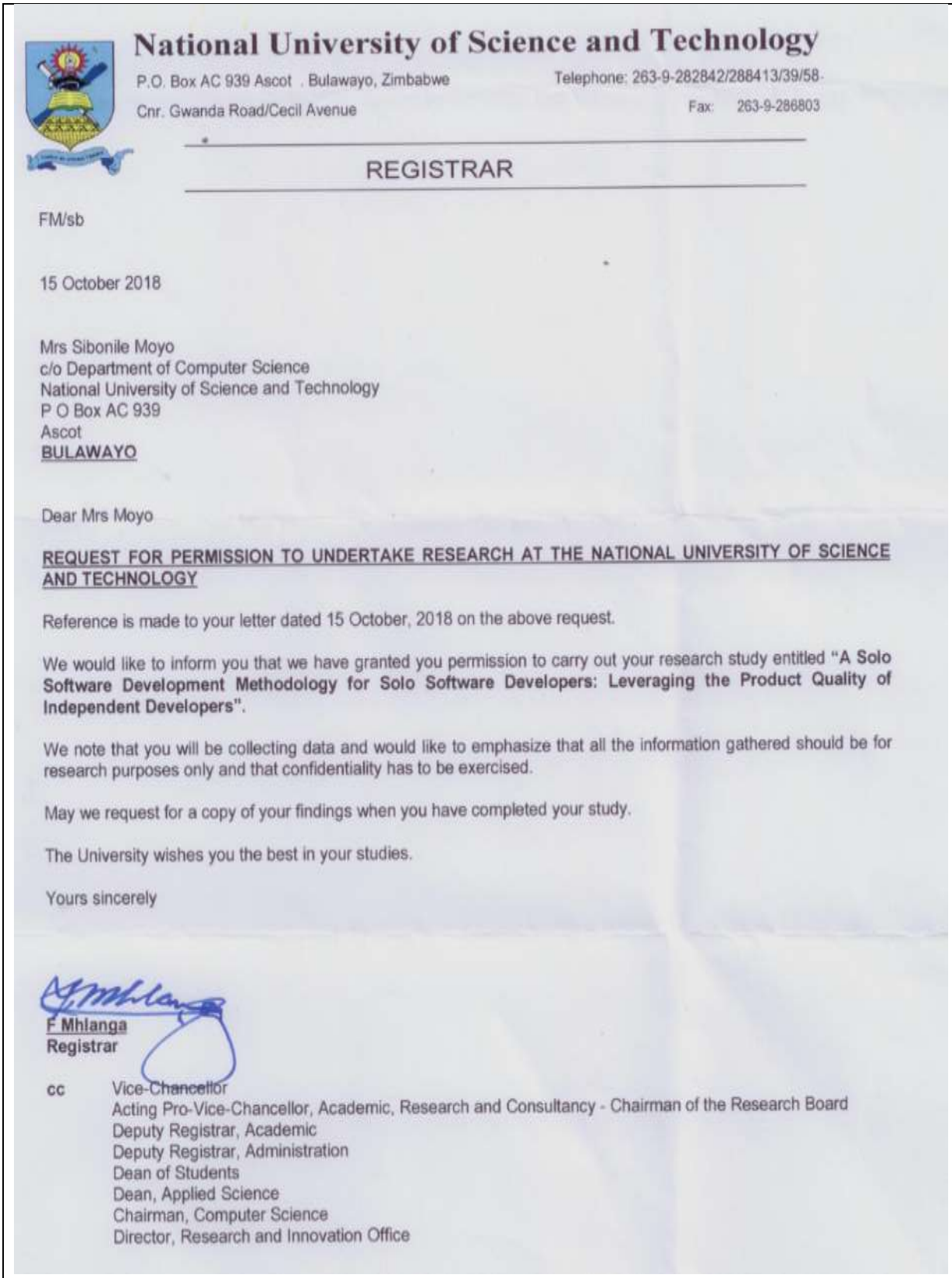
A Software Development Methodology for Solo Software Developers: Leveraging the Product Quality of Independent Developers

Qualification: PhD in Computer Science

Thank you for the application for research ethics clearance by the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee for the above-mentioned research. Ethics approval is granted for a period of five years, from 23 November 2018 to 23 November 2023.

1. The researcher will ensure that the research project adheres to the values and principles expressed in the UNISA Policy on Research Ethics.
2. Any adverse circumstance arising in the undertaking of the research project that is relevant to the ethicality of the study, as well as changes in the methodology, should be communicated in writing to the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee. An amended application could

Appendix A2. NUST Gate Keeper Clearance letter



National University of Science and Technology

P.O. Box AC 939 Ascot . Bulawayo, Zimbabwe
Cnr. Gwanda Road/Cecil Avenue

Telephone: 263-9-282842/288413/39/58
Fax: 263-9-286803

REGISTRAR

FM/sb

15 October 2018

Mrs Sibonile Moyo
c/o Department of Computer Science
National University of Science and Technology
P O Box AC 939
Ascot
BULAWAYO

Dear Mrs Moyo

REQUEST FOR PERMISSION TO UNDERTAKE RESEARCH AT THE NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

Reference is made to your letter dated 15 October, 2018 on the above request.

We would like to inform you that we have granted you permission to carry out your research study entitled "**A Solo Software Development Methodology for Solo Software Developers: Leveraging the Product Quality of Independent Developers**".

We note that you will be collecting data and would like to emphasize that all the information gathered should be for research purposes only and that confidentiality has to be exercised.

May we request for a copy of your findings when you have completed your study.

The University wishes you the best in your studies.

Yours sincerely

F Mhlanga
Registrar

cc Vice-Chancellor
Acting Pro-Vice-Chancellor, Academic, Research and Consultancy - Chairman of the Research Board
Deputy Registrar, Academic
Deputy Registrar, Administration
Dean of Students
Dean, Applied Science
Chairman, Computer Science
Director, Research and Innovation Office

Appendix B. Focus Group discussion guide

Time 2hrs

Having used the Secure-SSDM to design and implement your software product, this discussion seeks to establish your views on the applicability of the methodology in building quality and secure software. Feel free to air out your views as observed during the project. Your views are important in the construction of a quality software development methodology.

Ground Rules: Introduction, formulation and adoption of ground rules **5 mins**

Every idea is important

No idea is meaningless

To contribute (or support) an idea, show by a raise of hand

Only the person given the platform talks

The discussion will be directed by a set of question questions, participants are free to seek clarification on any issues pertaining to the discussion.

General Comments on the Methodology **40 mins**

1. Would you consider the Secure-SSDM to be a solution to a real problem/need in the solo software development environment currently?
2. Would you rate the practices embedded in the methodology adequate to build quality and secure software, if not what would you add?
3. How easy to follow are the practices in the Secure-SSDM? Which practices would you consider helpful, and which would you consider to be not?
4. Did you at any point feel you were asked to do more than just developing software?
5. What available tools would you suggest to ease the development process at any of the methodology stages?
6. What practices in the Secure-SSDM would you consider to be key in developing quality and secure software?
7. What improvements would you add to the methodology if you were given the opportunity to?
8. Would you consider using the Secure-SSDM in your future projects?

9. Would you recommend the methodology to any fellow developers?
10. Do you think the Secure-SSDM can be used to develop any kind of software system?

Phase by phase analysis: **30 mins**

What can you say about the adequacy of practices in these phases? -

i. Management-buy-in and standards adoption
ii. Requirements elicitation
iii. Sprint planning
iv. Development with code review
v. Sprint review and close
vi. Evaluation

Suggestions for Improvement ***30 mins***

i. Did you make any changes to any activity while performing a certain task?
ii. Did you make some provisions to perform a certain task because it was not clear how you were supposed to build a certain deliverable?
iii. If you were to add some activities to the methodology what would you add?
iv. Which activities do you think are core for the methodology?
v. What activities do you think are not necessary?

Appendix C Interview Guide – Industry developers

Title: A Software Development Methodology for Solo Software Developers: Leveraging the Product Quality of Independent Developers

Interview Guide **Time: 1 hour**

1. Introduction *5 minutes*

i. Researcher introduction

ii. Explanation of the case study, and what information is of interest, assuring the interviewee of anonymity.

iii. Explanation of the use of the data being collected, indicating the possibility of generating some publications from the data.

2. Comment on the Methodology as a whole *20 minutes*

i. Would you consider the Secure-SSDM to be a solution to a real problem/need in the solo software development environment currently?

ii. Would you rate the practices embedded in the methodology adequate to build quality and secure software, if not what would you add?

iii. How easy to follow are the practices in the Secure-SSDM? Which practices would you consider helpful, and which would you consider to be not?

iv. Did you at any point feel you were asked to do more than just developing software?

v. What available tools would you suggest to ease the development process at any of the methodology stages?

vi. What practices in the Secure-SSDM would you consider to be key in developing quality and secure software?

vii. What improvements would you add to the methodology if you were given the opportunity to?

viii. Would you consider using the Secure-SSDM in your future projects?

ix. Would you recommend the methodology to any fellow developers?

x. Do you think the Secure-SSDM can be used to develop any kind of software system?

Phase by phase analysis: *20 minutes*

i. What can you say about the adequacy of practices in these phases?

ii. Management-buy-in and standards adoption

iii. Requirements elicitation

iv. Release & sprint planning

v. Development review

vi. Sprint review and close

vii. Evaluation

3. Suggestions for improvement *10 minutes*

i. Did you make any changes to any activity while performing a certain task?

ii. Did you make some provisions to perform a certain task because it was not clear how you were supposed to build a certain deliverable?

iii. If you were to add some activities to the methodology what would you add?

iv. What activities do you think are not necessary?

4. Comment on the Quality of the Delivered Software product *3 minutes*

i. Does your Web-application have the required functionality?

ii. Do you think your component modules can be used to build applications in future projects?

iii. How did you test the security of your system?

iv. How secure is your system?

5. Conclusion *2 minute***s**

i. What else can you say about the methodology and its intended audience*?*

ii. Thank the interviewee and promise to give feedback on the interview once data transcription is complete.

Appendix D Focus Group discussion data capture template

Focus group data capturing template, adapted from Nili, Tate and Johnstone (2017)

Time **⇩**	**Moderator** **Question:**	**Participant Responses**									
		A	B	C	D	E	F	G	H	I	J